UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Urban Jurca

Razvoj aplikacije za prometno obveščanje

DIPLOMSKO DELO
UNIVERZITETNI ŠTUDIJSKI PROGRAM RAČUNALNIŠTVO
IN INFORMATIKA

MENTOR: prof. dr. Marko Bajec

Ljubljana, 2014

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Urban Jurca, z vpisno številko 63080025, sem avtor diplomskega dela z naslovom:

Razvoj aplikacije za prometno obveščanje

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marko Bajec
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki ”Dela FRI”.

V Ljubljani, dne Podpis avtorja:

Iskreno se zahvaljujem svojemu mentorju, prof. dr. Marko Bajcu, za hitre in izčrpne odgovore, ter vsesplošno pomoč pri izdelavi diplomskega dela.

Zahvaljujem se Janezu Sterletu in ostalemu kolektivu LTFE, ker ste mi omogočili, da sem lahko diplomo pisal ob službi, ter sem in tja spustil kakšen Spirent test.

Navsezadnje se zahvaljujem družini in prijateljem za vso podporo, pomoč in ideje. Rad vas imam.

Kazalo

Kazalo 9
Povzetek 11
Abstract 12
1. Uvod 13
2. Sorodna dela in primerjava 15
2.1. Razmere na cestah 15
2.2. Prometoid 15
2.3. Waze 16
2.4. Aplikacija Moja Cesta 17
3. Analiza in načrtovanje 19
4. Podatkovni viri 23
4.1. Viri podatkov 23
4.2. REST aplikacijski vmesnik in dobra praksa 23
4.3. Zbirka podatkov 25
4.4. Zbiranje prometnih podatkov s strani Dars 27
4.5. Zbiranje podatkov preko storitve Google+ 31
4.6. Aplikacijski vmesnik razvit za aplikacijo Moja Cesta 31
4.6.1. Splošni GET zahtevek po vseh prometnih dogodkih 31
4.6.2. GET zahtevek specifičnega uporabnika po prometnih dogodkih 32
4.6.3. Zahtevek POST za pošiljanje dogodkov iz mobilne aplikacije 33
4.6.4. Zahtevek PUT za spreminjane uporabniških nastavitev obveščanja 34
5. Aplikacija Moja Cesta 37
5.1. Spletna aplikacija 37
5.1.1. PHP ogrodje Laravel 37
5.1.2. Google+ Sign In 40
5.1.3. Geolokacija HTML5 in spreminjanje koordinat GPS v naslov 41
5.1.4. Asinhrono nalaganje podatkov 42

5.1.5. Pregled spletne aplikacije Moja Cesta 43
5.2. Mobilna Aplikacija 45
5.2.1. Obveščanje o prometnih dogodkih 45
5.2.2. Pridobivanje trenutne lokacije uporabnika in spreminjanje koordinat GPS v naslove ulic 48
5.2.3. Obveščanje o dogodkih 51
5.2.4. Dodajanje dogodkov 53
5.2.5. Asinhrono upravljanje podatkov in varnost 55
6. Možne razširitve 57
7. Zaključek 59
8. Viri 61

Povzetek

Ljudje radi vidimo, da je optimiziran vsak vidik našega življenja. To počnemo, da si lahko olajšamo vsakodnevna opravila in hkrati prihranimo čas, denar ali pa druge za nas pomembne stvari. Eno najpogostejših vsakodnevnih opravil, ki ga opravljamo skoraj vsi, je vožnja od doma do šole, službe ali kakšne druge končne destinacije, in nazaj. Če bi izračunali količino časa, ki ga preživimo v vozilu, bi kaj kmalu ugotovili, da ne gre za majno številko. Zakaj torej ne bi optimizirali vožnje in se izognili morebitnim pastem na poti, si s tem prihranili čas in se hkrati izognili stresu? S to idejo smo v sklopu diplomskega dela razvili mobilno in spletno aplikacijo Moja Cesta, ki uporabnike opozarja o dogodkih na cesti pred njimi. Uporabnik lahko pregleduje vse aktualne dogodke, v primeru, da odkrije neobjavljen dogodek, pa lahko le tega tudi prijavi in s tem opozori ostale uporabnike aplikacije. Ker lahko aplikacija sledi točni lokaciji uporabnika, ga z veliko natančnostjo opozarja o bližnjih dogodkih. Seveda smo želeli tudi, da si lahko uporabnik sam nastavi vse parametre obveščanja. S pomočjo aplikacije uporabnikom smo olajšali vsakodnevno vožnjo.

Ključne besede: DARS, promet.si, Android aplikacija, spletna aplikacija, spletne storitve

Abstract

People usually want every aspect of their lives optimized. We tend do this because we want to make our daily chores easier and at the same time save on time, money or anything else we find important. An example of this would be our daily commute to school, work or any other destination and back again. If we were to calculate the amount of time spent driving, we would quickly be able to conclude that it is not a small sum. Why not optimize our drive, avoid traffic events along the way and thus save on time and avoid the unnecessary stress? With this idea in mind, we developed a mobile and web application called *Moja Cesta* that alerts users about traffic events ahead. Users can view all relevant traffic events and can also submit events if a new one happened and is not yet listed. The new submitted event is then distributed to all other users. Because the application uses the users' exact location, it can alert the user about upcoming events with a very high degree of accuracy. We also wanted the user to be able to customize every alert parameter of the application. We wanted to make the users' daily commute easier and hassle free; hopefully, we achieved just that.

Key words: DARS, promet.si, Android application, web application, web services

1. Uvod

V Sloveniji je bilo do konca leta 2012 registriranih nekaj čez 1 066 000 osebnih avtomobilov, vse skupaj pa malo več kot 1 350 000 motornih vozil. Glede na trenutno število prebivalcev Slovenije imamo v Sloveniji na en avto na vsaka dva prebivalca. Če pogledamo še trenutno demografsko piramido vidimo, da je približno 1 500 000 prebivalcev starih med 18 in 65 let [4], torej aktivnih prebivalcev Slovenije. Če privzamemo, da je to skupina, ki ima v lasti vsa registrirana vozila, imamo v Sloveniji en avtomobil na 1,5 aktivnega prebivalca.

Dnevno sicer vsa registrirana vozila niso na cesti, vendar število vozil in voznikov, ki se vsak dan vozijo po cestah, nikakor ni zanemarljivo. Gostota prometa je najverjetneje večja zjutraj, ko ljudje prihajajo v službo, in popoldan, ko gredo iz službe, promet pa je bolj zgoščen tudi ob vikendih. Večja gostota prometa pomeni manjšo hitrost vožnje in daljši čas potovanja, ki pa je v današnjem hitrem tempu življenja postal veliko breme. Če k temu prištejemo tudi naravne dogodke, kontrole in nesreče, ki še dodatno ovirajo promet, se čas potovanja še toliko bolj podaljša. To poveča živčnost voznikih in morebitnih sopotnikih, kar pa posledično pomeni, da so manj pripravljeni na dnevne izzive.

Čakanju zaradi zastoja na cestah bi se lahko izognili, če bi pogledali podatke, ki jih ima na razpolago Dars o trenutnem stanju na cestah, in temu primerno spremenili svojo pot do cilja. Ljudje pa smo bitja navade in večino svojih dnevnih potovanj opravimo po isti poti v veri, da se na poti ne bo zgodil nepričakovan dogodek.

Iz zgornjega se je razvila ideja o aplikaciji, ki bi voznika predčasno opozarjala na morebitne dogodke na njegovi poti in vozniku omogočila, da prilagodi svojo pot oz. jo dogodku primerno spremeni. Hkrati bi lahko uporabnik predčasno pregledal trenutne prometne razmere in prilagodil pot vožnje. Podatke o dogodkih naj bi aplikacija pridobivala iz Darsove baze trenutnih cestnih dogodkov, hkrati pa bi uporabniki aplikacije lahko sami generirali dogodke in tako obveščali ostale uporabnike o aktualnih razmerah na slovenskih cestah. Aplikacija naj bi tekla na pametnih telefonih z operacijskim sistemom Android in v spletnem brskalniku, tako omogočala shranjevanje uporabniških preferenc dogodkov in obveščanja na njih.

Diplomsko delo je sestavljeno iz sedmih sklopov. Prvi sklop opisuje sorodna oziroma podobna dela in kako se ta dela dotikajo problematike obveščanja o prometnih dogodkih. Predstavili bomo mobilno aplikacijo Prometoid, Razmere na cestah in Waze.

V drugem sklopu bomo predstavili analizo aplikacije, načrt aplikacije in kaj vse je potrebno implementirati.

V tretjem sklopu bomo predstavili podatkovne vire z načinom obdelave in shranjevanja. Opisali bomo tudi spletne vmesnike, ki omogočajo prejemanje in oddajanje informacij in opisali način delovanja spletnih vmesnikov.

Četrti sklop je namenjen predstavitvi mobilne in spletne aplikacije. Predstavili bomo načrtovanje in razvoj mobilne aplikacije, ter izzive ki so se pojavili tekom razvoja. Predstavljene bodo tudi možne nadgradnje trenutne aplikacije, ki so se pojavile med testiranjem. Nato bomo predstavili spletno aplikacijo, v katero želimo prenesti večino funkcionalnosti mobilne aplikacije. Opisali bomo tudi ogrodje in relativno nepoznana dodatna orodja, ki smo jih uporabili pri razvoju spletne aplikacije.

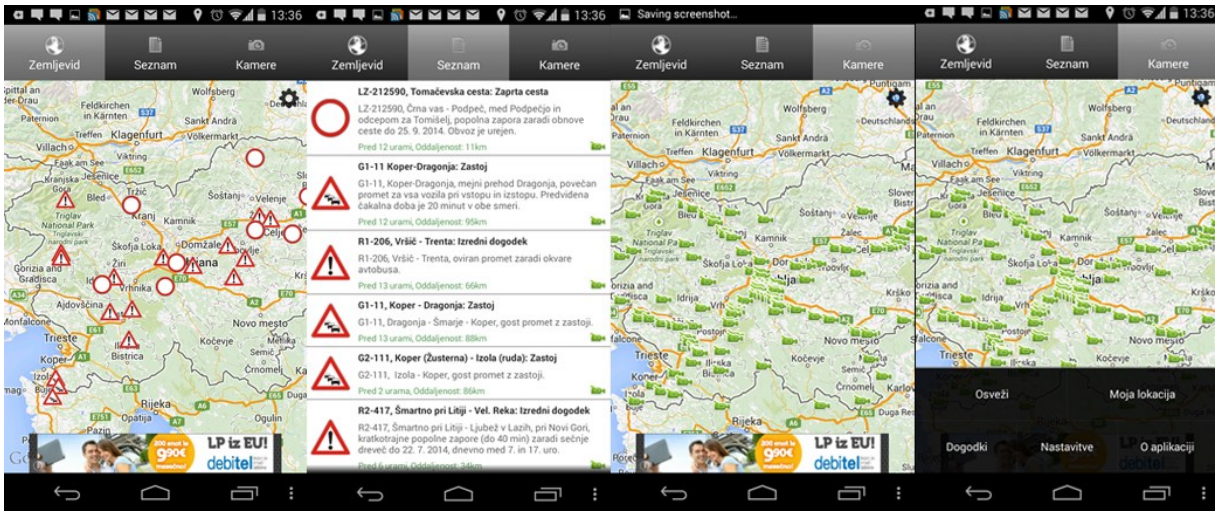
Peti sklop je namenjen ugotovitvam in možnim razširitvam aplikacije. Tukaj bomo opisali vse potencialne dodatne funkcionalnosti, ki jih bomo morda implementirali v aplikacijo Moja Cesta.

V zadnjem sklopu bomo predstavili zaključke in ugotovitve, ki so nastali med pisanjem te diplomske naloge.

2. Sorodna dela in primerjava

2.1. Razmere na cestah

Aplikacija Razmere na cestah (slika 1) je narejena po vzoru spletnega portala http://promet.si, vsi uporabljeni podatki pa so tudi pridobljeni iz omenjenega portala. Aplikacija uporablja seznamski prikaz dogodkov, prikaz dogodkov na zemljevidu in prikaz kamer na zemljevidu. Uporabnik si lahko nastavi urnik opozarjanja, na podlagi katerega ga aplikacija obvešča o novih dogodkih. Prav tako si lahko uporabnik nastavi regijo obveščanja o dogodkih.

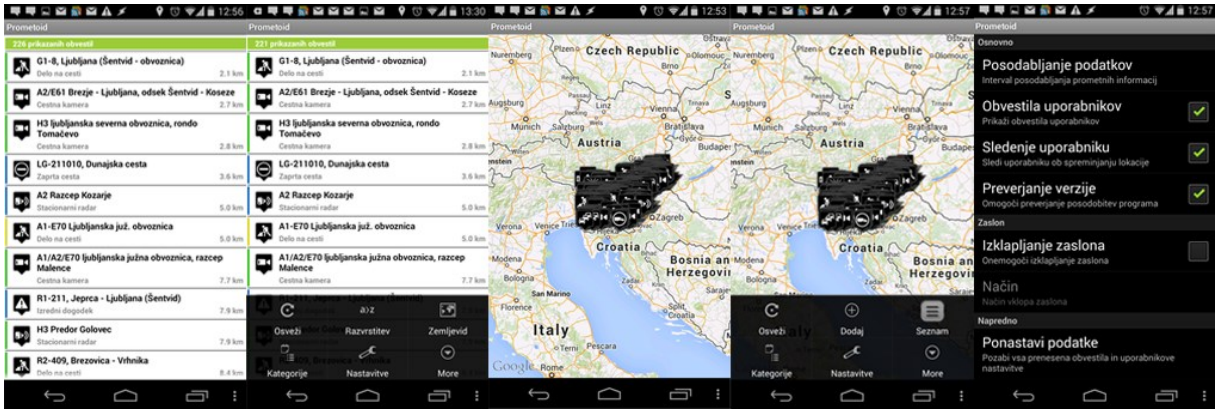


Slika 1: Razmere na cestah

Aplikacija ne ponuja možnosti obveščanja uporabnika o dogodkih glede na njegovo trenutno lokacijo. Uporabniki ne morejo ustvarjati svojih dogodkov, kar pomeni, da je ažurnost podatkov odvisna od Darsa, ki skrbi za dodajanje novih dogodkov in od katerega črpa dogodke spletna storitev http://promet.si. Trenutne možnosti izbora obveščanja so preveč omejujoče, saj uporabnik lahko prejme preveč ali premalo informacij o stanju na cestah, glede na njegov izbor obveščanja.

2.2. Prometoid

Je aplikacija, napisana za operacijski sistem Android, ki omogoča hiter in preprost pregled nad stanjem na slovenskih cestah (slika 2). Informacije prikaže na seznamu in Google Maps zemljevidu. Omogoča tudi vpogled v lokacijo prometnih kamer. Ker sledi uporabnikovi lokaciji, lahko sortira dogodke po oddaljenosti. Uporabniki prav tako lahko dodajajo svoje dogodke. Aplikacija ima možnost nastavitve intervala osveževanja dogodkov, tako da so le-ti dokaj ažurni. Aplikacija je prosto dosegljiva na aplikacijski trgovini Play Store.

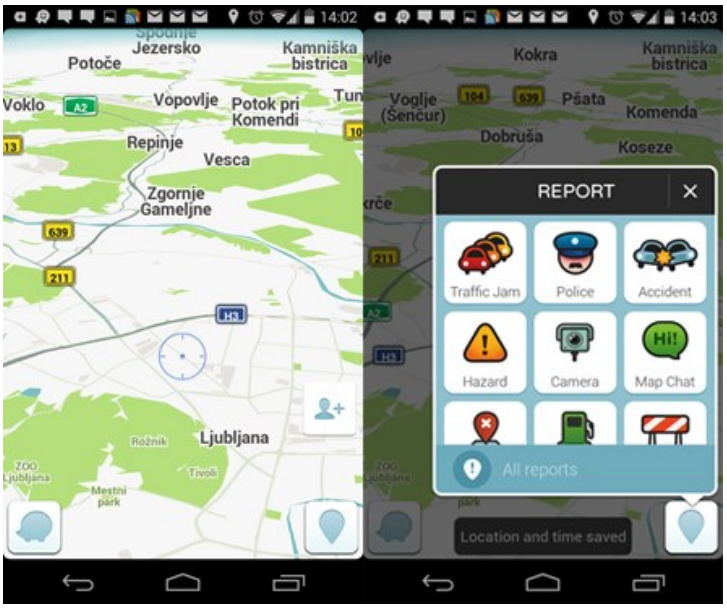


Slika 2: Prometoid

Aplikacija je nastala leta 2010 in od takrat ni bila veliko posodobljena. Pomanjkljivosti so predvsem na področju uporabniškega vmesnika in navigacije po aplikaciji, ki je uporabniku neprijazna. Uporabniki lahko tudi dodajajo svoje dogodke, vendar je proces okoren in zamuden; če predpostavljamo, da uporabniki uporabljajo aplikacijo med vožnjo, je prav tako tudi potencialno nevaren.

2.3. Waze

Waze (slika 3) je ena izmed največjih navigacijskih aplikacij, ki ponujajo tudi prometne informacije. Aplikacija temelji na skupnosti, preko katere uporabniki »gradijo« zemljevid in obveščajo ostale uporabnike o trenutnih prometnih razmerah. Aplikacija je na voljo za Android, iOS in Windows Phone platformo. Poleg mobilne aplikacije je na voljo tudi spletna aplikacija, ki se nahaja na https://www.waze.com/livemap. Uporabniki lahko dodajajo poročila o dogodkih in s tem obvestijo ostale uporabnike.



Slika 3: Waze

Aplikacija je zelo intuitivna in prijazna do uporabnika, vendar hiter pregled nad trenutnimi poročili pokaže, da slovenski uporabniki ne uporabljajo te funkcionalnosti.

2.4. Aplikacija Moja Cesta

Na podlagi že razvitih aplikacij smo želeli uporabiti njihove dobre lastnosti in hkrati odstraniti neuporabne ali odvečne dele. Odločili smo se, da mora aplikacija pridobivati podatke z Darsovega informacijskega sistema in na enostaven in hiter način omogočiti uporabniku dodajanje svojih dogodkov. Postopek dodajanja novega dogodka mora biti za uporabnika intuitiven, hiter in enostaven.

Ključna funkcionalnost aplikacije je tudi ta, da storitev obveščanja o dogodkih teče v ozadju, zato ni potrebno, da da ima uporabnik aplikacijo odprto na ekranu. Uporabnik zažene aplikacijo, ta pa deluje v ozadju, medtem ko uporabnik naprej normalno uporablja svojo mobilno napravo. V primeru, da se uporabnik približa dogodku, ga aplikacija na le-tega opozori. Uporabniku smo želeli omogočiti tudi prijavo v mobilno in spletno aplikacijo preko Google+ Sign In funkcionalnosti, ter mu omogočiti, da se njegove shranjene preference uveljavijo na vseh instancah aplikacije, ne glede na platformo. Kot dodano vrednost smo želeli uvesti tudi t.i. *gamification*, kjer uporabniki med seboj tekmujejo in se borijo za mesto najboljšega obveščevalca. Menimo, da bo tekmovalni aspekt aplikacije pozitivno vplival na njen sprejem in hitro razširitev.

3. Analiza in načrtovanje

Želeli smo razviti aplikacijo, ki bo na voljo na več različnih platformah – spletno in mobilno aplikacijo. Aplikacija mora biti sposobna pobirati ažurne podatke iz Darsovega informacijskega sistema in hkrati ponujati možnost prejemanja dogodkov. Aplikacija mora avtomatično vsakih nekaj minut prenesti podatke z spletnega aplikacijskega vmesnika, ki se nahaja na http://kazipot1.promet.si. Prejete podatke v obliki XML mora aplikacija primerno obdelati in shraniti v bazo na strežniku. Iz prejetih podatkov je treba primerno obdelati GPS koordinate dogodka z ostalimi metapodatki in jih iz oblike XML pretvoriti v elemente v relacijski podatkovni bazi. Aplikacija mora ponuditi svoj spletni aplikacijski vmesnik, preko katerega lahko spletna in mobilna aplikacija dostopata do podatkov. Prejemanje in pošiljanje podatkov preko aplikacijskega vmesnika mora potekati v skladu z arhitekturo REST, podatki pa morajo biti v obliki JSON.

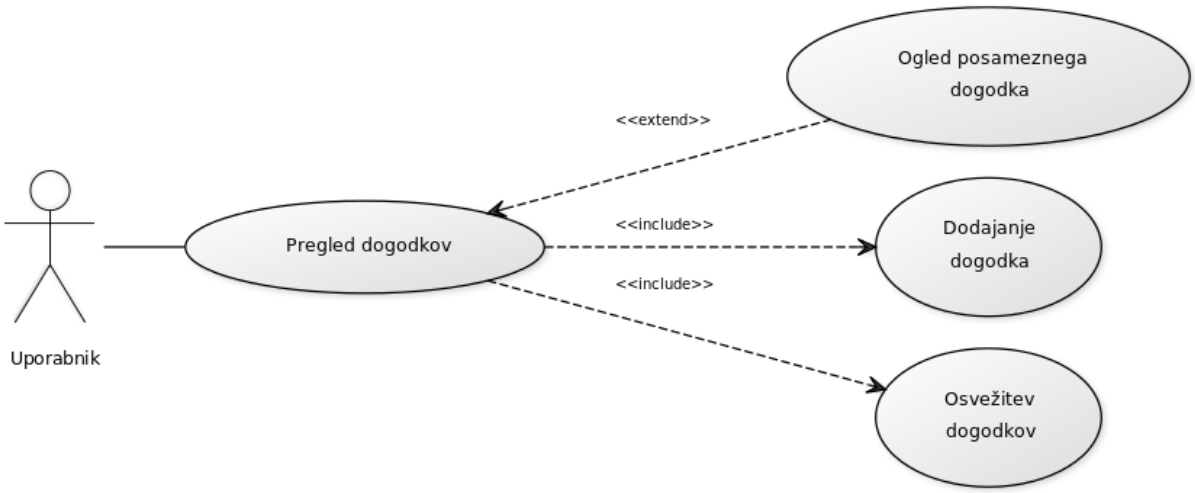
Aplikacija mora imeti implementirano funkcijo prijave uporabnika in beleženje uporabnikovih aktivnosti. Uporabniku želimo ponuditi igralniško izkušnjo s tem, da primerjamo njegove aktivnosti z ostalimi uporabniki. V ta namen mora aplikacija beležiti in shraniti število prejetih dogodkov uporabnikov. Enotno prijavo uporabnika implementiramo z uporabo Google+ Sign In knjižnice, kjer se uporabi Google poverilnice uporabnika za prijavo v aplikacijo. Poleg prijave uporabnika moramo omogočiti personalizacijo aplikacije tako, da uporabniku omogočimo shranjevanje nastavitev obveščanja, poskrbeti pa moramo tudi za konsistentno uveljavljanje teh nastavitev preko vse platform. Prometne dogodke moramo prikazati tako na zemljevidu, z uporabo Google Maps aplikacijskih vmesnikov, kot v obliki seznama na spletni in mobilni platformi. Poleg položaja dogodkov želimo slediti lokaciji uporabnika in mu primerno prikazati njegovo lokacijo glede na ostale dogodke. Lokacijo uporabnika želimo čim bolj natančno določiti v vseh platformah. V primeru, da nam platforma to dopušča, želimo uporabnika o bližnjih dogodkih tudi obvestiti. Obveščanje naj poteka z uporabo vibriranja in spuščanja zvokov.

Sliki 4 in 5 prikazujeta diagrama primerov uporabe za spletno in mobilno aplikacijo, s katerimi želimo pokazati interakcijo med akterji in sistemom Moja Cesta. Iz diagramov je razvidno, da je mobilna aplikacija le razširitev spletne aplikacije. Aplikacija zajema naslednje primere uporabe:

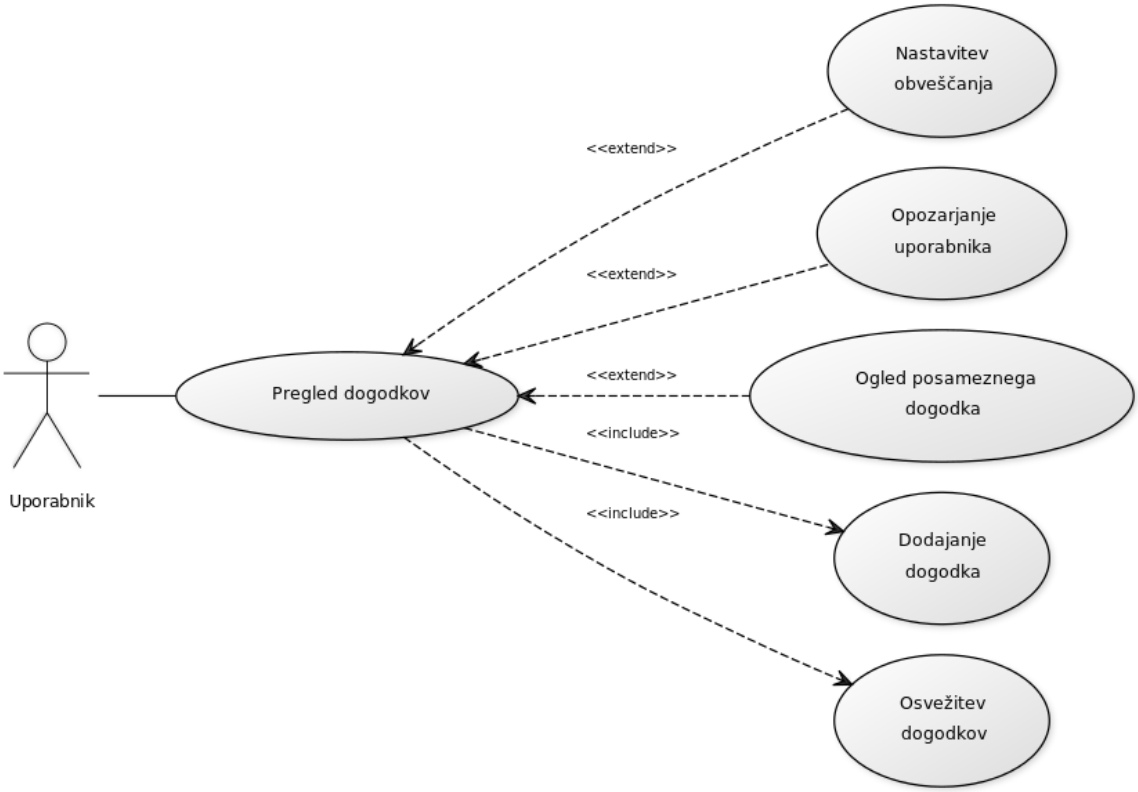
- Pregled dogodkov: ob zagonu aplikacije se poišče uporabnikovo trenutno lokacijo in prenese prometne dogodke iz strežnika ter jih prikaže uporabniku. Dogodki se lahko sortirajo po razdalji oziroma po času nastanka, odvisno od platforme uporabe. Ob zadostnem premiku uporabnika se dogodki posodobijo na mobilni aplikaciji. Po

pretečenem določenem časovnem intervalu se dogodki ponovno prenesejo iz strežnika in osvežijo. Primer uporabe se zaključi, ko uporabnik zapre aplikacijo.

- Osvežitev dogodkov: dogodki se osvežijo ob kliku na namenski gumb za osvežitev znotraj aplikacije. Ob kliku se s strežnika ponovno prenesejo dogodki. Ob koncu prenosa se primer uporabe konča, uporabniku pa se prikaže pregled dogodkov.
- Dodajanje dogodkov: dogodki se dodajo, ko uporabnik klikne na namenski gumb za izbor dogodka in izbere želeni dogodek. Ob kliku se pošlje podatek na strežnik, kjer se shrani v bazo dogodkov. Po pošiljanju podatka se osvežijo podatki, primer uporabe pa se zaključi. Uporabniku se prikaže pregled dogodkov.
- Ogled posameznega dogodka: ob kliku na posamezni dogodek iz seznama dogodkov se odpre nov pogled s približanim dogodkom in njegovim opisom. Primer uporabe se zaključi, ko uporabnik spremeni pogled.
- Opozarjanje uporabnika: v kolikor ima uporabnik nastavljeno opozarjanje in se približa na definirano razdaljo kateremu od dogodkov in ima vklopljeno funkcijo vibriranja ali opozarjanja z zvokom, aplikacija uporabnika na dogodek opozori. Uporabnika se opozori le enkrat, dokler je v radiju opozarjanja za dogodek. Ko se uporabnik premakne izven radija opozarjanja, se primer uporabe zaključi.
- Nastavitve obveščanja: uporabnik lahko spreminja nastavitve aplikacije. Izbere želene dogodke, za katere želi prejemati podatke in določi parametre obveščanja. Ko uporabnik shrani nastavitve, se sproži osvežitev dogodkov in uporabnika vrnemo na pregled dogodkov.



Slika 4: Diagram primera uporabe za spletno aplikacijo



Slika 5: Diagram primera uporabe za mobilno aplikacijo

4. Podatkovni viri

Aplikacija Moje Cesta uporablja veliko različnih podatkov za svoje delovanje. Podatki so pridobljeni preko javno dostopnih virov. Poleg zbiranja podatkov smo morali tudi implementirati funkcionalnost deljenja in manipuliranja s podatki preko spletnega aplikacijskega vmesnika.

4.1. Viri podatkov

Glede na zastavljene funkcionalnosti aplikacije potrebujemo več virov podatkov. Prvi vir podatkov je pridobljen preko Darsovega informacijskega sistema obveščanja – podatke avtomatsko zbere in obdela Dars, mi pa jih pridobimo prek njega.. Ti podatki so zanesljivi in ažurni do te mere, kolikor hitro podatke posodablja Dars. Drugi vir podatkov so podatki, ki jih generirajo uporabniki sami preko mobilne aplikacije, skupaj z vsemi relevantnimi meta podatki. Glede na vir podatka moramo le-te obdelati in sprejeti na različne načine. Torej: podatke objavljene s strani Darsa moramo na nek načini dobiti in obdelati, medtem ko moramo za uporabniške podatke ponuditi aplikacijski vmesnik, preko katerega lahko aplikacija pošlje in prejme podatke.

Uporabljamo tudi podatke dobljene s storitve Google+, saj smo tako v spletni in mobilni aplikaciji omogočili vpis v aplikacijo z uporabniškimi Google+ poverilnicami. Z dostopom do njih lahko dobimo dostop do uporabniških podatkov, kot so uporabnikova ime in priimek uporabnika, uporabnikova profilna slika ipd.

4.2. REST aplikacijski vmesnik in dobra praksa

Kratica REST je okrajšava za Representational State Transfer. Nanaša se na brez-stansko, klient-strežnik komunikacijo. V skoraj vseh primerih se za delovanje uporablja protokol HTTP. REST je arhitekturni stil oziroma pristop pri snovanju spletnih aplikacij. Namesto uporabe bolj kompleksnih mehanizmov, kot so CORBA, RPC ali SOPA, se za povezavo med napravami uporabljajo preprosti klici HTTP.

Specifikacija HTTP/1.0 definira metode GET, POST in HEAD; specifikacija HTTP/1.1 definira metode OPTIONS, PUT, DELETE, TRACE in CONNECT. Čeprav specifikaciji definirata 8 metod, pa so v praksi pri snovanju aplikacijskih vmesnikov REST uporabljane le metode GET, POST, PUT in DELETE. Uporaba teh metod ali stavkov nam omogoča manipulacijo s spletnimi viri. Omenjene metode se lahko preslikajo 1:1 na osnovne stavke SQL (tabela 1) za upravljanje s podatki v podatkovnih zbirkah ali metode CRUD (Create, Read, Update in Delete).

Operacija	SQL	HTTP
Ustvari	INSERT	POST
Preberi	SELECT	GET
Posodobi	UPDATE	PUT
Izbriši	DELETE	DELETE

Tabela 1: primerjava stavkov HTTP in SQL

Pri snovanju aplikacijskih vmesnikov REST želimo implementirati intuitiven in razvijalcu enostaven vmesnik, ki se ga z lahkoto razume in uporablja. Da bi vmesnik čim bolj poenostavili, je naš cilj razviti dobre naslove spletnih vmesnikov in pravilno uporabo stavkov HTTP. Načelo dobre prakse je, da v naslovnih aplikacijskih vmesnikov uporabljamo samostalnike. Ti naj se nanašajo na vire, s katerimi želimo manipulirati. Prav tako želimo zmanjšati število naslovov URL na vir – naš cilj je, da ima vsak vir maksimalno dva naslova URL [3], [13].

Za primer vzemimo, da imamo spletno aplikacijo, ki hrani podatke o psih. Do podatkov lahko dostopamo preko aplikacijskega vmesnika, ki ima definirana dva naslova URL:

1. http://www.mojipsi.si/psi
2. http://www.mojipsi.si/psi/1234

Prvi naslov se nanaša na celotno zbirko psov, drugi pa na specifičnega psa. Namesto, da bi definirali več naslovov URL za manipulacijo s podatki o psih, raje uporabimo stavke HTTP (tabela 2).

Vir	POST create	GET read	PUT update	DELETE delete
/psi	Ustvari novega psa	Vrni vse pse	Posodobi vse pse	Izbriši vse pse
/psi/1234	Napaka	Vrni psa z id=1234	Posodobi psa z id=1234	Izbriši psa z id=1234

Tabela 2: Primer uporabe HTTP stavkov

V kolikor želimo izvajati bolj zapleteno manipulacijo nad viri, lahko dodatne parametre dodamo na koncu naslova URL za znakom ?. Če od aplikacijskega vmesnika želimo dobiti vse pse, ki se trenutno nahajajo v parku, lahko ustvarimo naslednjo zahtevo:

GET http://mojipsi.si/psi?lokacija=park

Seveda moramo ustrezno obravnavno zahtevka implementirati tudi v aplikacijskem vmesniku.

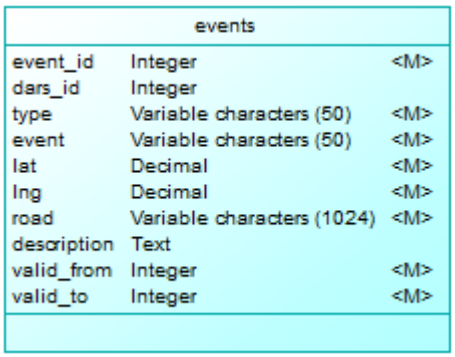
4.3. Zbirka podatkov

Vse uporabne in obdelane podatke želimo shraniti v podatkovno zbirko SQL. V podatkovno zbirko shranjujemo vse aktualne prometne dogodke in podatke o uporabnikih skupaj z njihovimi preferencami. Z uvedbo podatkovne zbirke ustvarimo centraliziran dostop do podatkov, z uporabo naprednih tehnologij, kot je *caching*, pa pospešimo dostop do njih.

Aplikacija Moja Cesta potrebuje 2 podatkovni tabeli:

- tabelo dogodkov, kjer hranimo dogodke in njihove lastnosti in
- tabelo uporabnikov, kjer hranimo uporabniške podatke in preference.

Tabelo dogodkov v podatkovni zbirki poimenujemo »events« (slika 6 in tabela 3); vsebuje vse relevantne podatke za hranjenje aktualnih prometnih dogodkov. Aktualni prometni dogodek je tisti, katerega veljavnost še ni potekla. V primeru da dogodku poteče veljavnost, je le-ta odstranjen iz tabele.



Slika 6: Tabela dogodkov events

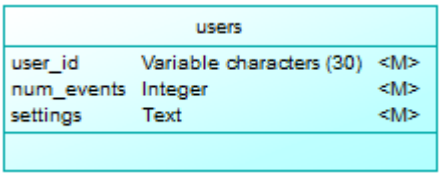
Atribut	Namen
Event_id	ID dogodka, kot ga definira aplikacija moja Cesta
Dars_id	ID dogodka, kot ga definira DARS
Type	Tip dogodka
Event	Ime dogodka
Lat	Geografska širina dogodka
Lng	Geografska dolžina dogodka
Road	Cesta na kateri se je zgodil dogodek
Description	Opis dogodka
Valid_from	Začetek veljavnosti dogodka v EPOCH formatu
Valid_to	Konec veljavnosti dogodka v EPOCH formatu

Tabela 3: Opis atributov v tabeli dogodkov

Veljavnost dogodkov zaradi večje učinkovitosti SQL raje zapišemo v numerični obliki EPOCH, ki šteje sekunde od 1.1.1970. Izkaže se, da iskanje po bazi SQL z uporabo stavkov SELECT vrne boljše časovne rezultate, ko se uporabljajo podatki tipa INT, kot pa ko so v uporabi podatki tipa TIMESTAMP ali DATETIME [5].

Tabela nalašč ni normalizirana do tretje normalne oblike. Vsebuje dva atributa, ki se med dodajanje novih dogodkov pojavita večkrat. Ta dva atributa sta type in events. Tabele nismo normalizirali do tretje normalne oblike zato, ker tabela vsebuje le aktualne prometne dogodke, ki pa jih je na dnevni ravni relativno malo. Posledično tabela nikoli ne zraste do te mere, da bi jo bilo potrebno zmanjšati. Normalizacija tabele tako ne prinese nobenih izboljšav v zmogljivosti.

Tabelo uporabnikov v podatkovni zbirki poimenujemo »users« in vsebuje nabor uporabniških podatkov (slika 7 in tabela 4).



Slika 7: Tabela uporabnikov users

Atribut	Namen
User_id	User_id pridobljen preko Google+
Num_events	Število dogodkov, ki jih je uporabnik poslal preko aplikacije Moja Cesta
Settings	Uporabnikove preference v JSON obliki

Tabela 4: Opis atributov v tabeli uporabnikov

Atribut *User_id* pridobimo preko storitve Google+, ko se uporabnik prvič vpiše v aplikacijo, bodisi mobilno ali spletno. Z uporabo tega atributa lahko dostopamo do Google+ uporabniških podatkov, kot so profilna slika in ime in priimek, ki jih uporabimo znotraj aplikacij.

V atributu *settings* hranimo uporabniške nastavitve za obveščanje o dogodkih v obliki JSON. Ko kreiramo zapis v bazi uporabnikov users, nastavimo tudi privzeto vrednost nastavitev obveščanja. Po privzetih nastavitvah je uporabnik obveščen o vseh dogodkih, nato pa lahko v nastavitvah poljubno spreminja obveščanje.

Primer nastavitev:

```
{
"zastoj":true,
"izredni_dogodek":true,
"delo_na_cesti":false,
"zaprta_cesta":true,
"poledica":true,
"veter":true,
"sneg":true,
"nesreca":true,
"radar":true
}
```

V kolikor je ob tipu dogodka navedeno *true*, bo uporabnik o dogodku obveščen. V nasprotnem primeru, se mu dogodki ne prenesejo na aplikacijo in o njih posledično ni obveščen.

4.4. Zbiranje prometnih podatkov s strani Dars

Prometne podatke pridobivamo preko vmesnika API spletne strani http://promet.si, ki vsebuje vse aktualne podatke pridobljene preko Darsovega informacijskega sistema. Spletni vmesnik je dosegljiv na http://kazipot1.promet.si/kazipot/services/dataexport/. Uvodna stran spletnega vmesnika ponuja dokumentacijo za pravilno obliko zahtevkov po prometnih podatkih (slika 8).

Slika 8: Dokumentacija aplikacijskega vmesnika promet.si

Po prebrani dokumentaciji ugotovimo, da se primeren format podatkov dobi s klicem sledečega spletnega vmesnika: http://kazipot1.promet.si/kazipot/services/dataexport/exportDogodki.ashx?format=XML&version=1.0.0&reportType=SHORT&language=SI&sortOrder=VELJAVNOSTODDESC&icons=DEFAULT&crsId=EPSG:4326.

V zahtevku smo definirali, da želimo, da:

1. nam spletni vmesnik vrne rezultat v obliki XML verzije 1.0 -> format=XML&version=1.0.0
2. dobimo strnjeno obliko rezultata -> reportType=SHORT
3. dobimo rezultat v slovenščini -> language=SI
4. so dogodki sortirani po veljavnosti padajoče -> sortOrder=VELJAVNOSTODDESC
5. želimo dobiti privzete ikone za dogodek -> icons=DEFAULT
6. želimo, da so koordinate GPS zakodirane po WGS 84 formatu -> crsId=EPSG:4326

Rezultat nam vrne pravilno formuliran odgovor XML, iz katerega izluščimo posamezne dogodke in za nas relevantne podatke.

Zahteva GET:
http://kazipot1.promet.si/kazipot/services/dataexport/exportDogodki.ashx?format=XML&version=1.0.0&reportType=SHORT&language=SI&sortOrder=VELJAVNOSTODDESC&icons=DEFAULT&crsId=EPSG:4326

Odgovor:


```
<?xml version="1.0" encoding="utf-8"?>
<dogodki version="1.0.0" currentDateTime="2014-07-21T12:29:49.6166742Z" lastUpdate="2014-07-21T12:27:30.307Z" crsId="EPSG:4326" periodFrom="2014-07-21T12:29:49.6010502Z" periodTo="2014-07-21T12:29:49.6010502Z" language="SI">
  <dogodek updated="2014-07-21T12:27:30.307Z">
    <point>14.8614077523756,46.5348662111093</point>
    <id>164396</id>
    <kategorija>R2</kategorija>
    <cesta>R2-425, Poljana - Šentvid</cesta>
    <vzrok>Izredni dogodek</vzrok>
    <opis>R2-425, Poljana - Šentvid, oviran promet zaradi razlitega olja.</opis>
    <veljavnostOd>2014-07-21T12:25:00Z</veljavnostOd>
    <veljavnostDo>2014-07-22T13:25:00Z</veljavnostDo>
    <icon>http://kazipot1.promet.si/kazipot/services/dataexport/icons/dogodki_v1/dogodek3.gif</icon>
  </dogodek>
  <dogodek updated="2014-07-21T12:07:58.303Z">
    <point>15.3613045106101,46.256964478476</point>
    <id>164395</id>
    <kategorija>A1</kategorija>
    <cesta>A1-E57 Maribor - Ljubljana</cesta>
    <vzrok>Delo na cesti</vzrok>
    <opis>Na avtocesti Maribor - Ljubljana med počivališčem Zima in priključkom Dramlje v smeri Maribora bo do 21. 7. 2014 zaradi del zaprt vozni pas.</opis>
    <veljavnostOd>2014-07-21T12:07:00Z</veljavnostOd>
```

```
  <veljavnostDo>2014-07-21T21:50:00Z</veljavnostDo>
  <icon>http://kazipot1.promet.si/kazipot/services/dataexport/icons/dogodki_v1/delo1.gif</icon>
 </dogodek>
…
```

Ker želimo podatke shraniti v podatkovno zbirko SQL, moramo rezultat primerno obdelati. Za obdelavo podatkov smo razvili skripto PHP, ki se na strežniku zažene vsakih pet minut. Skripta v bazi dogodkov izbriše vse dogodke, ki je pretekla veljavnost. Nato prenese najnovejše podatke, preveri, če dogodki v pridobljenem rezultatu že obstajajo v bazi, in jih primerno ažurira. Pomembno je tudi, da posebne znake pravilno kodiramo, saj se s tem zavarujemo pred napačnim prikazom na različnih platformah. Pretvorimo vse znake, ki so specifični za slovenski jezik – »ČčŠšŽž« v entitete HTML po sledeči tabeli (tabela 5):

Znak	HTML entiteta
Č	Č
č	č
Š	Š
š	š
Ž	Ž
ž	ž

Tabela 5: Entitete HTML za znake, ki so specifični v slovenskem jeziku

Izločimo tudi dogodke, ki so specifični za tovornjake.

Glavni del skrite PHP, ki skrbi za prenos in hrambo ažurnih podatkov s portala promet.si:

```
$query = 'DELETE FROM events WHERE event_id IN (SELECT * FROM (
   (SELECT event_id FROM events WHERE valid_to < ' . time() . ')
      ) AS p)';
   mysqli_query($dbc, $query) or die(print(mysqli_error($dbc) . $query));

   $xml =
simplexml_load_file("http://kazipot1.promet.si/kazipot/services/dataexport/exportDogodki.ashx?format
=XML&version=1.0.0&reportType=SHORT&language=SI&sortOrder=VELJAVNOSTODDESC&ico
ns=DEFAULT&crsId=EPSG:4326");

   foreach ($xml as $dogodek) {

      $id = $dogodek->id;
      $point = explode(',', $dogodek->point);
      $lat = $point[1];
      $lng = $point[0];
      $event = clean_slovenian_strings($dogodek->vzrok);
      $tip = str_replace('č', 'c', str_replace(' ', '_', strtolower($dogodek->vzrok)));
      if ($event == "Prepoved za tovornjake") {
         continue;
```

```
        }
        $cesta = clean_slovenian_strings($dogodek->cesta);
        $opis = clean_slovenian_strings($dogodek->opis);
        $od = strtotime($dogodek->veljavnostOd);
        $do = strtotime($dogodek->veljavnostDo);
        $query = 'SELECT * FROM events WHERE Dars_id=' . $id;
        $result = mysqli_query($dbc, $query) or die(print(mysqli_error($dbc) . $query));
        if (mysqli_num_rows($result) > 0) {
           continue;
        }

        $query = 'INSERT INTO events
        (Dars_id, type, event, lat, lng, road, description, valid_from, valid_to)
        VALUES (' . $id . ',"' . $tip . '","' . $event . '",' . $lat . ',' . $lng . ',"' . $cesta . '","' . $opis . '",' . $od . ','
. $do . ')';
        mysqli_query($dbc, $query) or die(print(mysqli_error($dbc) . $query));
      }
```

Da lahko skripta teče na vsakih 5 minut, jo je na strežniku potrebno vključiti v proces CRON, ki je časovni razporejevalnik opravil v operacijskih sistemih UNIX. Ključna komponenta procesa CRON je konfiguracijska datoteka crontab, v katero vključimo potrebne parametre za periodični zagon procesov.

Do crontab dostopamo z ukazom »crontab –e«, kar pomeni, da bi radi urejali crontab datoteko. Primer crontab datoteke za skripto PHP, ki skrbi za ažuriranje podatkov:

```
5 * * * * php /usr/local/bin/save_Dars_events.php
```

Vsaka vrstica datoteke crontab se v grobem loči na dva dela: na del, kjer definiramo urnik zagona procesa in drugi del, kjer definiramo, kaj želimo, da se zažene.

```
# * * * * *  ukaz, ki naj se zažene
# ┬ ┬ ┬ ┬ ┬
# │ │ │ │ │
# │ │ │ │ │
# │ │ │ │ └──── dan v tednu (0 - 6) (0 do 6 so nedelja do sobota)
# │ │ │ └─────────── mesec (1 - 12)
# │ │ └────────────────── dan v mesecu (1 - 31)
# │ └──────────────────────── ura (0 - 23)
# └─────────────────────────────── minuta (0 - 59)
```

Naš ukaz zažene skripto PHP vsakih pet minut vsako uro, na vsak dan v vsakem mesecu, skozi celo leto.

4.5. Zbiranje podatkov preko storitve Google+

Z storitvijo Google+ Sign In omogočimo uporabniku enovit vpis v aplikacijo Moja Cesta. Ko se uporabnik vpiše v aplikacijo preko Sign In funkcionalnosti, dobi aplikacija OAuth žeton, ki ga lahko uporabimo v imenu uporabnika in z njim posledično izboljšamo uporabniško izkušnjo aplikacije. Storitev Google+ Sign In lahko implementiramo v spletnih aplikacijah in aplikacijah razvitih na platformi iOS in Android [2].

Uporabniške podatke lahko pridobimo z uporabo aplikacijskega vmesnika HTTP, ki ga ponuja storitev Google+ Sign In. Da pridobimo podatke o specifičnem uporabniku, moramo uporabiti metodo People:Get. Metodo kličemo na sledeči način:

GET https://www.googleapis.com/plus/v1/people/*userId*

Primer rezultata:


```
{
 "kind": "plus#person",
 "id": "118051310819094153327",
 "displayName": "Chirag Shah",
 "url": "https://plus.google.com/118051310819094153327",
 "image": {
  "url": "https://lh5.googleusercontent.com/-
XnZDEoiF09Y/AAAAAAAAAAI/AAAAAAAAYCI/7fow4a2UTMU/photo.jpg"
 }
}
```


User_id uporabnika pridobimo, ko se uporabnik vpiše v aplikacijo preko Sign In storitve in ta podatek shranimo v bazo users. Nato ga uporabimo, da preko vmesnika pridobimo podatke o profilni sliki uporabnika – image->url in o njegovem imenu in priimku – displayName.

4.6. Aplikacijski vmesnik razvit za aplikacijo Moja Cesta

Za komunikacijo med strežnikom in mobilno ter spletno aplikacijo je bilo potrebno razviti svoj aplikacijski vmesnik REST, katerega lahko uporabi aplikacija. Držali smo se priporočenih smernic pri razvoju vmesnika in uporabili standardne metode HTTP za komunikacijo med aplikacijo in strežnikom:

1. zahteva HTTP GET – od strežnika zahtevamo podatke,
2. zahteva HTTP POST – na strežnik naložimo podatke,
3. zahteva HTTP PUT – na strežniku spremenimo podatke.

4.6.1. Splošni GET zahtevek po vseh prometnih dogodkih

Če aplikacija izvede splošno zahtevo GET, ji strežnik vrne vse aktualne prometne dogodke v obliki JSON iz podatkovne baze events na strežniku. Klic je namenjen uporabniku aplikacije, ki ni prijavljen preko storitve Google+ in posledično nima definiranih svojih preferenc prometnega obveščanja.

Primer:

GET http://www.mojacesta.com/events

Odgovor:


```
{
"events":[
        {
                "event_id":"15857",
                "type":"nesreca",
                "event":"Nesre\u010da",
                "lat":"46.0588606232919",
                "lng":"14.4541889593367",
                "description":"Na ljubljanski zahodni obvoznici med razcepom Koseze in
                priklju\u010dkom Lj. - Brdo v smeri Kozarij je zaradi prometne nesre\u010de zaprt
                vozni pas.",
                "road":"A2-E61 Ljubljanska zah. obvoznica"
        },
        {
                "event_id":"15858",
                "type":"delo_na_cesti",
                "event":"Delo na cesti",
                "lat":"46.3701684304591",
                "lng":"14.1591430357196",
                "description":"Na avtocesti Karavanke - Ljubljana med priklju\u010dkom Lesce in
                priklju\u010dkom Lipce v smeri Karavank\/Avstrije bo do 22. 7. 2014 zaradi del
                oviran promet.",
                "road":"A2-E61 Karavanke - Ljubljana"
        },
…
```


4.6.2. GET zahtevek specifičnega uporabnika po prometnih dogodkih

Če aplikacija doda zahtevku GET še user_id, pridobljen preko storitve Google+ Sign In, lahko uporabniku serviramo le tiste prometne dogodke, za katere si je uredil nastavitve obveščanja. Poleg prometnih dogodkov pošlje strežnik tudi trenutne nastavitve obveščanja. Nastavitve se pošljejo vsakič znova, saj se lahko v vmesnem času spremenijo na drugi platformi. Pošljemo tudi statistiko uporabnikovega pošiljanja – število poslanih dogodkov in njegovo mesto po lestvici pridnih pošiljateljev.

Primer:

GET http://www.mojacesta.com/events/106895082351762346801

Odgovor:


```
{
"user_stats":{
        "num_events":"52",
        "rank":"1"
},
"preferences":{
        "zastoj":true,
        "izredni_dogodek":true,
        "delo_na_cesti":false,
        "zaprta_cesta":true,
        "poledica":true,
        "veter":true,
        "sneg":true,
        "nesreca":true,
        "radar":true
},
"events":[
        {
                "event_id":"15857",
                "type":"nesreca",
                "event":"Nesre\u010da",
                "lat":"46.0588606232919",
                "lng":"14.4541889593367",
                "description":"Na ljubljanski zahodni obvoznici med razcepom Koseze in
                priklju\u010dkom Lj. - Brdo v smeri Kozarij je zaradi prometne nesre\u010de zaprt
                vozni pas.",
                "road":"A2-E61 Ljubljanska zah. obvoznica"
        },
…
```


4.6.3. Zahtevek POST za pošiljanje dogodkov iz mobilne aplikacije

Zahtevek POST je namenjen za prejemanju prometnih dogodkov iz mobilne aplikacije, ki jih generirajo prijavljeni uporabniki. Po prejetju zahtevka POST se iz njega izluščijo poslane informacije, nad njimi pa naredimo obdelavo. Preden shranimo prejet prometni dogodek, pogledamo po obstoječi bazi prometnih dogodkov – events –, če v radiju 100m že obstaja isti tip dogodka. V kolikor tak dogodek že obstaja, mu podaljšamo veljavnost in prejeti dogodek zavržemo.

Preden zaključimo obdelavo zahtevka POST, moramo uporabnika, ki je poslal prometni dogodek, nadgraditi tako, da mu povečamo število poslanih dogodkov za 1 in mu tako morebiti izboljšamo rang med uporabniki.

Stavek SQL, ki smo ga uporabili za ugotavljanje dogodkov v radiju 100m po obstoječi bazi, je sledeč:

```
SELECT event_id, ( 6371 * acos( cos( radians(RECIEVED_LATITUDE) ) * cos( radians( lat ) ) * cos(
radians( lng ) - radians(RECIEVED_LONGITUDE) ) + sin( radians(RECIEVED_LATITUDE) ) * sin(
radians( lat ) ) ) ) AS distance
   FROM events
         WHERE type = RECIEVED_TYPE
   HAVING distance < 0.1
   ORDER BY distance LIMIT 1
```

Primer:

POST http://www.mojacesta.com/events

Spremenljivke, ki so zahtevane v zahtevku POST:

1. User_id -> user_id, dobljen preko Google+ Sign in storitve
2. Event -> tip poslanega dogodka
3. Lat -> geografska širina
4. Lng -> geografska dolžina
5. Road -> Ime ceste, na kateri se je zgodil dogodek

Odgovor:

HTTP 200 OK

4.6.4. Zahtevek PUT za spreminjane uporabniških nastavitev obveščanja

Zahtevek PUT je namenjen posodabljanju uporabniških nastavitev obveščanja o prometnih dogodkih. Ker forme HTML (do HTML različice 4 in XHTML različice 1) ne podpirajo metode PUT, moramo uporabiti zvijačo. Naredimo običajno formo HTML, ki uporablja metodo POST, vendar dodamo skrivno polje, v katerega vpišemo metodo pošiljanja:

```
<form class="form-inline" id="settingsForm" action="">
   <legend>Prikaži:</legend>
   <label for="zastoj" class="checkbox">
      Zastoj <input id="zastoj" name="zastoj" type="checkbox" />
   </label>
   <label for="izredni_dogodek" class="checkbox">
      Izredni dogodek <input id="izredni_dogodek" name="izredni_dogodek"
type="checkbox"/>
   </label>
   <label for="delo_na_cesti" class="checkbox">
      Delo na cesti <input id="delo_na_cesti" name="delo_na_cesti" type="checkbox" />
   </label>
   <label for="zaprta_cesta" class="checkbox">
      Zaprta cesta <input id="zaprta_cesta" name="zaprta_cesta" type="checkbox" />
   </label>
   <label for="poledica" class="checkbox">
      Poledica <input id="poledica" name="poledica" type="checkbox" />
   </label>
   <label for="veter" class="checkbox">
      Veter <input id="veter" name="veter" type="checkbox" />
   </label>
```

```
    <label for="sneg" class="checkbox">
       Sneg <input id="sneg"name="sneg" type="checkbox" />
    </label>
    <label for="nesreca" class="checkbox">
       Nesreca <input id="nesreca"name="nesreca" type="checkbox" />
    </label>
    <label for="radar" class="checkbox">
       Radar <input id="radar" name="radar" type="checkbox" />
    </label>
    <input type="hidden" name="_method" value="PUT"/>
    <button id="settingsButton" type="submit" class="btn">Vredu</button>
</form>
```

Primer:

PUT http://www.mojacesta.com/users/{user_id}

Odgovor:

HTTP 200 OK

5. Aplikacija Moja Cesta

Razvili smo spletno aplikacijo Moja Cesta in mobilno različico aplikacije za platformo Android. V prihodnosti želimo razviti tudi iOS različico aplikacije.

5.1. Spletna aplikacija

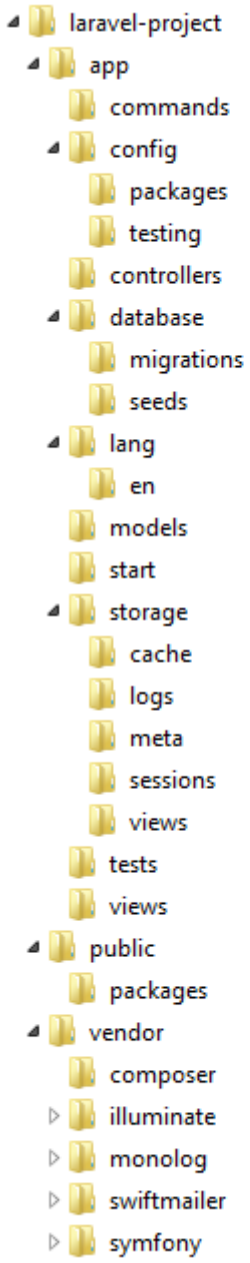
Spletna aplikacija Moja Cesta je razvita z namenom pregledovanja prometnih dogodkov na uporabniku bolj prijazen način, kot ga ponujajo obstoječe aplikacije, ki smo jih že omenili v drugem poglavju. Poleg pregledovanja smo omogočili uporabniku možnost personalizacije rezultatov obveščanja o prometnih dogodkih z uporabo storitve Google+ Sign In, preko katere uporabnika in njegove nastavitve shranimo za bodočo rabo. Aplikacija je razvita na strežniku LAMP; je skupek odprtokodne programske opreme, ki skupaj tvori delujoč spletni strežnik. Sestavljen je iz operacijskega sistema Linux, spletnega strežnika Apache, podatkovne baze MySQL in skriptnega jezika PHP.

5.1.1. PHP ogrodje Laravel

Za spletno aplikacijo smo izbrali Laravel 4, odprtokodno ogrodje PHP. Laravel je definiran kot »full stack framework« oziroma ogrodje s popolnim skladom, saj opravlja vse funkcije od spletnega serviranja do upravljanja s podatkovno bazo in generiranja dokumentov HTML. Razvijalec upravlja ogrodje preko ukazne vrstice orodja Artisan, ki generira in upravlja projektno okolje Laravel. Artisan se uporablja za generacijo ogrodne kode in shem podatkovnih zbirk, skrbi pa tudi za migracijo shem podatkovnih zbirk do upravljanja virov in konfiguracij [10].

Uporabljen moto ogrodja Laravel je »Convention over configuration«. Za razliko od ostalih podobnih ogrodij, ki potrebujejo veliko dodatne konfiguracije XML, je Laravel ne potrebuje skoraj nič. Zaradi tega je struktura vseh aplikacij, ustvarjenih z ogrodjem Laravel, zelo podobna in hitro prepoznavna za razvijalca. Ogrodje Laravel posledično vsili razvijalcu uporabo samodejno generirane datotečne strukture (slika 9).

Glavni direktoriji so app, public in vendor. Direktorij app vsebuje kontrolorje, model, poglede in sredstva aplikacije. Večina razvite kode se nahaja temu direktoriju. Direktorij public vsebuje vse javno dostopne vire, tudi zagonsko datoteko index.php, ki zažene spletno aplikacijo. V tem direktoriju se tudi nahajajo ostali javno dostopni viri, kot so datoteke CSS, datoteke JavaScript, slike in ostalo. Direktorij vendro vsebuje kodo razvito s strani tretjih oseb. V tipični Laravelovi aplikaciji bo direktorij vseboval Laravelovo izvorno kodo, njene odvisnosti in morebitne vtičnike in njihovo kodo.



Slika 9: Datotečna struktura projekta narejenega v ogrodju Laravel

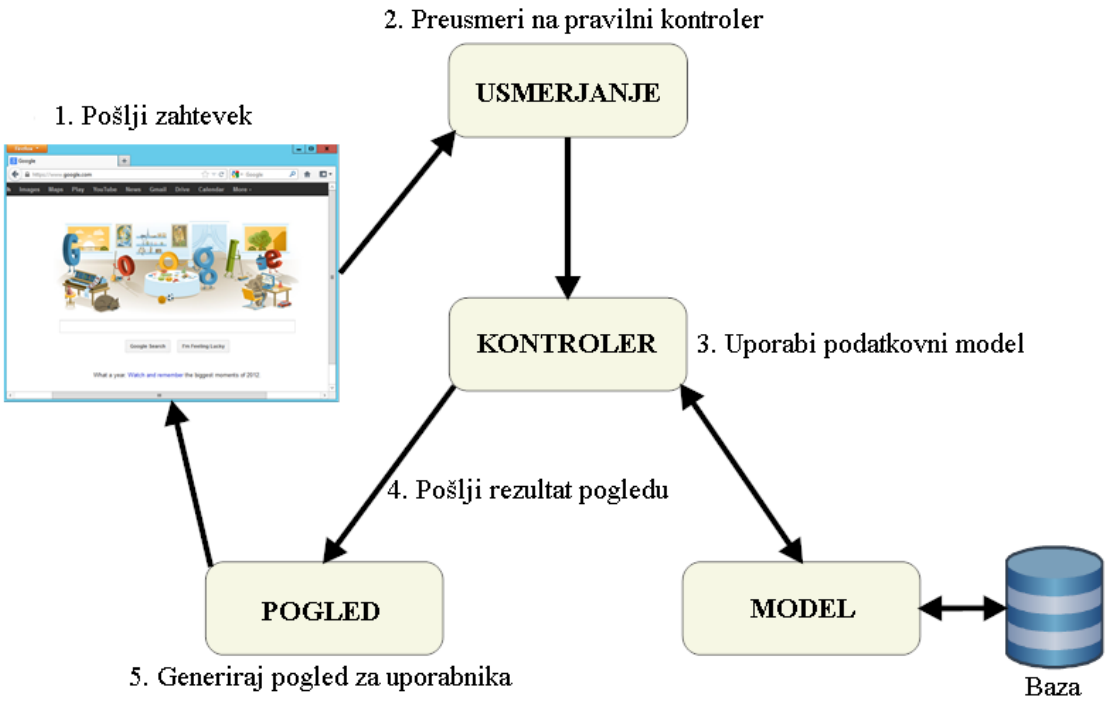
Ogrodje Laravel strogo sledi arhitekturnemu pristopu MVC, kjer strogo ločimo poslovno logiko aplikacije od vhodne in predstavitvene logike, povezane z grafičnim vmesnikom aplikacije. Aplikacija MVC je sestavljena iz treh komponent (slika 10):

1. Model – je domena okoli katere se razvije aplikacijo. Modeli so osnovani na entitetah iz realnega sveta, kot na primer oseba, osebni račun ali izdelek. Če gradimo blog, so modeli bloga komentar in članek. Modeli so večinoma obstojni in jih shranjujemo zunaj aplikacije, običajno v podatkovni bazi. Vendar je model več kot samo podatek ali skupek podatkov – z modelom lahko uveljavljamo omejitve nad podatki. Če imamo spletno trgovino, lahko definiramo v modelu naročila, (npr.: v primeru da je nakup

manjši od 10€, na ta nakup ne moremo uveljavljati dodatnega popusta). Tako imamo to omejitev na ravni podatkovnega modela, omejitve tudi ne moremo obiti na ravni aplikacije.

2. Pogled – je vizualna predstavitev podatkovnega modela z dodanim kontekstom. Po navadi je to odgovor, ki ga ogrodje ponudi spletnemu brskalniku – koda HTML. Pogled je zadolžen za ustvarjanje grafičnega vmesnika aplikacije, ki je osnovan na podatkih v modelu. Čeprav pogled predstavi vmesnik uporabniku, je obdelava vnesenih podatkov v vmesnik izven domene pogleda. Ko se podatki prikažejo uporabniki, je delo pogleda zaključeno.

3. Kontroler – je koordinator, ki povezuje model s pogledom. Kontroler je zadolžen za procesiranje prejetih podatkov, manipulacijo nad podatkovnim modelom ter določanjem akcij, ki naj bi se zgodile; to so npr. prikaz pogleda ali preusmeritev na drugo stran.



Slika 10: Povezava med MVC komponentami

Brskalnik pošlje zahtevek Laravel spletni aplikaciji, ki ga obdela Laravel usmerjevalna komponenta. Na podlagi prejetega zahtevka vzorca URL, se Laravel usmerjevalnik odloči, kateri metodi, definirani v kontrolerju, se pošlje zahtevek. Kontrolerski razred lahko takoj vrne odgovor brskalniku v obliki pogleda, vendar se v večini dinamičnih spletnih straneh najprej

zgodi interakcija s podatkovnim modelom, ki je element PHP, ki predstavlja nek element aplikacije in je odgovoren za komunikacijo z bazo. Po klicu podatkovnega modela vrne kontroler končni pogled in celotno spletno stran nazaj brskalniku.

5.1.2. Google+ Sign In

Aplikacijo Moja Cesta smo želeli prirediti potrebam uporabnikov. V ta namen smo vključili funkcionalnost Google+ Sign In, preko katere si lahko vsak posameznik prilagodi aplikacijo. Platforma Google+ omogoča razvijalcem dostop do Googlovih storitev, ko so preverjanje uporabnika, namestitev aplikacij OTA, uporaba interaktivnih člankov in socialnih omrežij [4].

Da smo lahko začeli uporabljati storitev Google+ Sign In, smo morali omogočiti Google+ API za svoj projekt v Google developer console (slika 11).

Slika 11: Vključen Google + API v Google developer console

Nato smo definirali, iz katerih URL naslovov lahko dostopamo do aplikacijskega vmesnika Google+ (slika 12).

Slika 12: Nastavitve za spletno aplikacijo

V aplikacijo je potrebno dodati še Google+ Sign In gumb in funkcijo JavaScript, ki skrbi za preverjanje z Googlovimi strežniki. Po uspešnem preverjanju želimo prijavni gumb skriti ter naložiti uporabniške podatke za spletno aplikacijo.

Koda HTML za gumb:

```
<span id="signinButton">
  <span
    class="g-signin"
    data-callback="signinCallback"
    data-clientid="CLIENT_ID"
    data-cookiepolicy="single_host_origin"
    data-requestvisibleactions="http://schema.org/AddAction"
    data-scope="https://www.googleapis.com/auth/plus.login">
  </span>
</span>
```

Potrebna funkcija JavaScript:

```
<script type="text/javascript">
    (function() {
     var po = document.createElement('script'); po.type = 'text/javascript'; po.async = true;
     po.src = 'https://apis.google.com/js/client:plusone.js';
     var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(po, s);
    })();

function signinCallback(authResult) {
      if (authResult['access_token']) {
        $('#login').hide();
        $('#userInfo').show();
        getUserInfo();
      } else if (authResult['error']) {
        console.log('There was an error: ' + authResult['error']);
      }
   }
</script>
```

5.1.3. Geolokacija HTML5 in spreminjanje koordinat GPS v naslov

Uporabniku smo želeli ponuditi možnost prikaza trenutne lokacije in ulice, na kateri se nahaja, na zemljevidu. V ta namen smo uporabili funkcionalnost HTML5, ki spletni aplikaciji omogoča pridobitev lokacije uporabnika brskalnika. Ko je funkcionalnost implementirana, bo brskalnik poslal lokalno mrežno informacijo lokacijskim storitvam Google, da pridobi približek lokacije. Lokalne mrežne informacije, ki jih uporablja lokacijska storitev Google, so vidne brezžične dostopne točke in njihova oddajna moč, informacija o lokalnem usmerjevalniku in naslov IP naprave. V primeru, da obstaja rezultat, bomo dobili koordinato GPS približne lokacije. Spletni brskalnik bo uporabnika vprašal za dovoljenje za pridobitev lokacije. Šele po eksplicitni potrditvi lahko brskalnik uporabi zmožnost pridobivanja lokacije [1].

Na podlagi prejete koordinate lahko naredimo obratno geokodo, kjer na podlagi koordinat pridobimo naslov ulice, na kateri se nahaja točka GPS. V ta namen uporabimo storitev Google GeoCode, ki nam vrne seznam ulic sortiranih po razdalji od trenutne točke.

Primer uporabe HTML5 Geolokacije in pridobivanja naslova ulice:

```
<!--Try W3C Geolocation (Preferred)-->
      if (navigator.geolocation) {
         navigator.geolocation.getCurrentPosition(function (position) {
            initialLocation = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
            map.setCenter(initialLocation);
            map.setZoom(12);

            var marker = new google.maps.Marker({
               position: initialLocation,
               map: map,
               title: 'Moja Lokacija',
               icon: 'img/icons/blue-dot.png'
            });

            reverseGeocode(initialLocation);
         }, function () {
            handleNoGeolocation();
         });
      }
function reverseGeocode(position){
      var addressUrl =
'https://maps.googleapis.com/maps/api/geocode/json?latlng='+position.lat()+','+position.lng()+'&sensor
=true';
      $.getJSON(addressUrl, function(data){
         $("#location").text(data['results'][0]['formatted_address']);
      });
   }
```

5.1.4. Asinhrono nalaganje podatkov

Z namenom pospešitve nalaganja spletne aplikacije smo za prenos vseh ne-statičnih vsebin uporabili asinhrono nalaganje. Uporabili smo tehnologijo AJAX – asinhron JavaScript in XML za zagotovitev asinhronega prenosa. Tehnologija AJAX se uporablja za ustvarjanje interaktivnih spletnih aplikacij in je značilna za aplikacije Web 2.0. Omogoča hitrejše in bolj tekoče posodabljanje vsebine na spletni strani, saj se podatki izmenjujejo s strežnikom na asinhron način in brez potrebe po ponovnem nalaganju spletne strani. Držalo smo se dobre prakse in vse skripte, ki izmenjujejo podatke, premaknili na dno dokumentov HTML, takoj za </body> oznako. Prav to nam omogoča, da celotno spletno stran naložimo hitreje, saj bi v nasprotnem primeru prenos skript blokiral prenos ostalih vidnih komponent spletne strani. Tako se skripte naložijo potem, ko je že prenesena celotna stran in naknadno napolnijo manjkajočo vsebino spletne strani.

Čeprav kratica AJAX namiguje izključno na uporabo JavaScript in XML tehnologij, lahko uporabimo tudi druge tehnologije. Za prenos podatkov med strežnikom in spletno ter mobilno aplikacijo uporabljamo zapis podatkov JSON. Razlog za to odločitev ni toliko v povezavi z zmogljivostjo, kot pa v povezavi z lažjo obdelavo prejetih podatkov, saj zahteva obdelava podatkov JSON manj kode kot pa obdelava podatkov XML, in lažjo berljivostjo podatkov. Za

asinhron prenos podatkov JSON uporabljamo funkcijo knjižnice JQuery $.getJSON., ki od strežnika zahteva prenos podatkov JSON.

Primer:

```
$.getJSON(dataUrl, function(data){
        displayData();
});
```

Za pošiljanje podatkov JSON proti strežniku uporabimo definirano funkcijo knjižnice JQuery $.ajax, kjer definiramo tip podatkov.

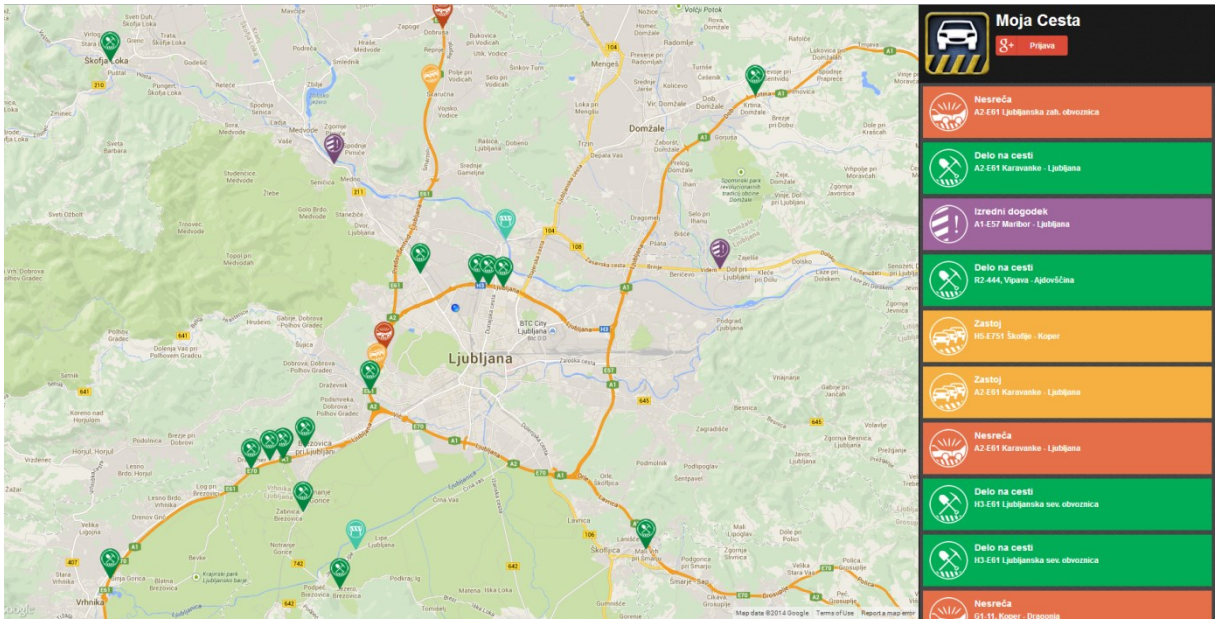
Primer:

```
$.ajax({
        type: 'POST',
        url: 'users/' + user_id,
        data: $('#settingsForm').serialize(),
        success: function(data){
           getDataForUser();
        }
    });
```

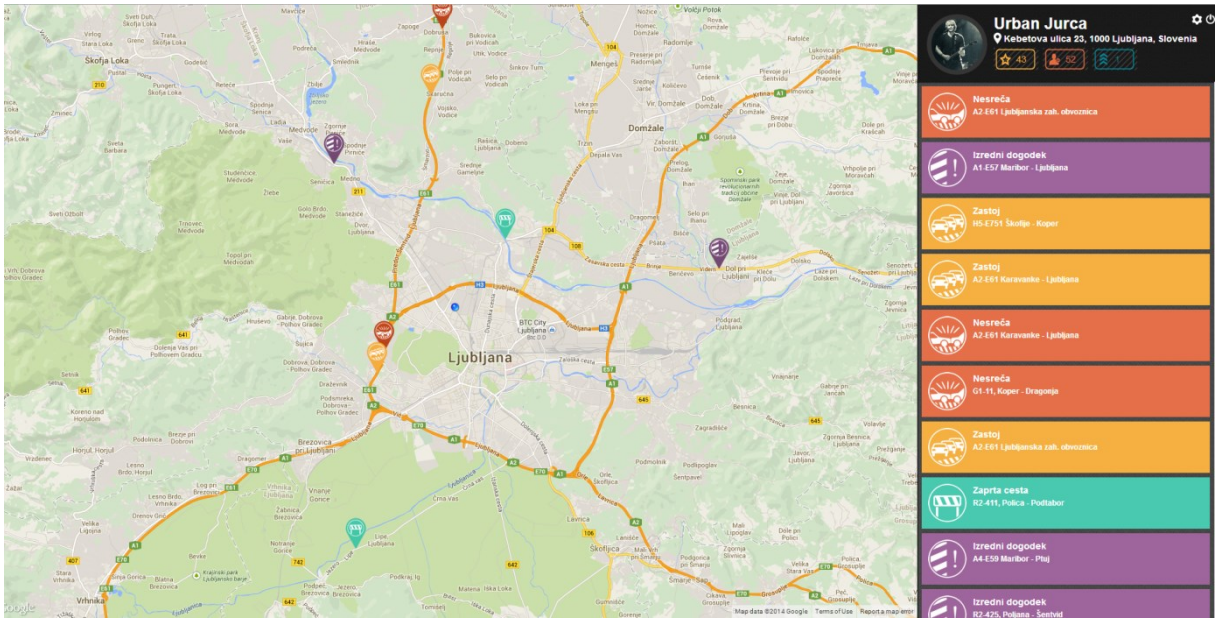
5.1.5. Pregled spletne aplikacije Moja Cesta

Končni rezultat uporabljenih spletnih tehnologij je spletna aplikacija Moja Cesta, ki je dosegljiva na naslovu http://mojacesta.com.

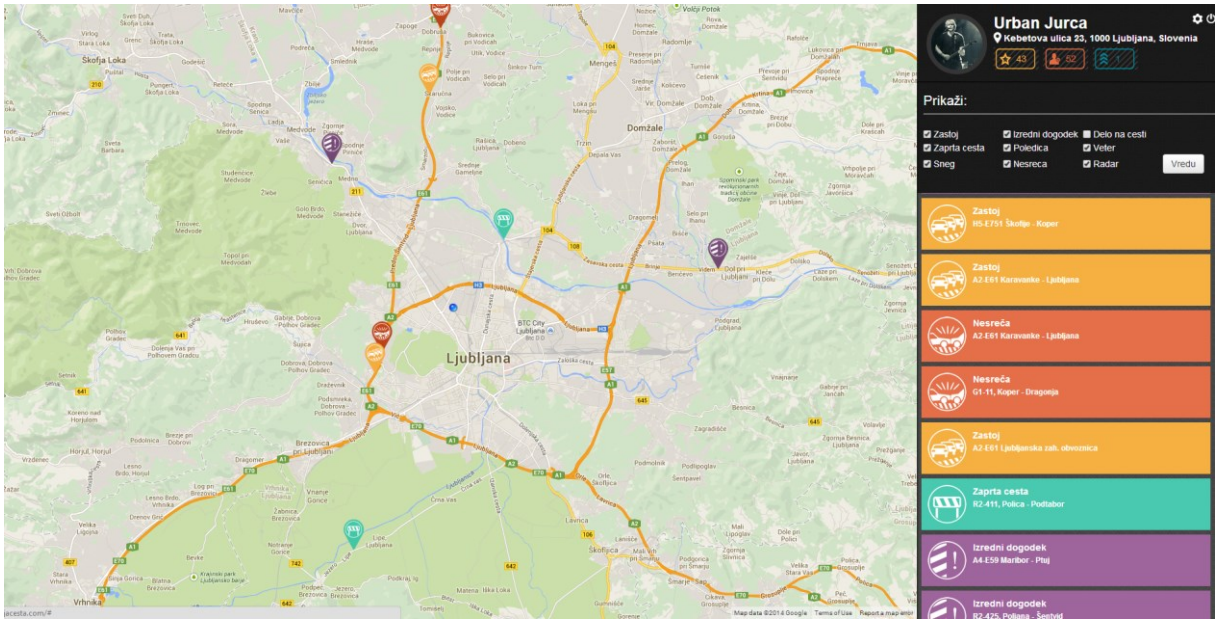
Uporabniku se prikaže začetna spletna stran, ki je sestavljena iz zemljevida, na kateri so prikazani vsi prometni dogodki, in seznama dogodkov na desni, ki je povezan z dogodki na zemljevidu (slika 13). Klik na dogodek iz seznama približa zemljevid na ta dogodek. Uporabniku je ob prvem obisku predstavljena možnost prijave v aplikacijo. Ko se uporabnik prijavi v aplikacijo, se zamenja ikona aplikacije z njegovo profilno sliko storitve Google+ in njegovim imenom in trenutnim naslovom, v kolikor je delil lokacijo z aplikacijo (slika 14). Prikažejo se tudi njegova statistika sodelovanja. Uporabniku se ponudi možnost spreminjanja nastavitev obveščanja (slika 15), kjer si izbere, kakšen tip dogodka ga zanima. Sprememba se shrani v bazo na strežniku.



Slika 13: Zgled strani z neprijavljenim uporabnikom



Slika 14: Izgled strani s prijavljenim uporabnikom



Slika 15: Možnost spreminjanja obveščanja za prijavljenega uporabnika

5.2. Mobilna Aplikacija

Mobilna aplikacija Moja Cesta je razvita z namenom aktivnega obveščanja uporabnika o prometnih dogodkih in nudi možnost, da uporabnik deli svoje izkušnje o prometnih dogodkih z ostalimi uporabniki aplikacije. Mobilno aplikacijo je bilo potrebno nadgraditi z funkcionalnostjo aktivnega obveščanja in možnostjo dodajanja novih dogodkov.

Glavna naloga, ki jo mora aplikacija opraviti, je obveščanje uporabnika na prometni dogodek v upoštevanju izbranih poljubnih nastavitev. Pomembno je tudi, da je aplikacija zmožna teči v ozadju aplikacijskega sklada, a še vedno obvestiti uporabnika o prometnih dogodkih.

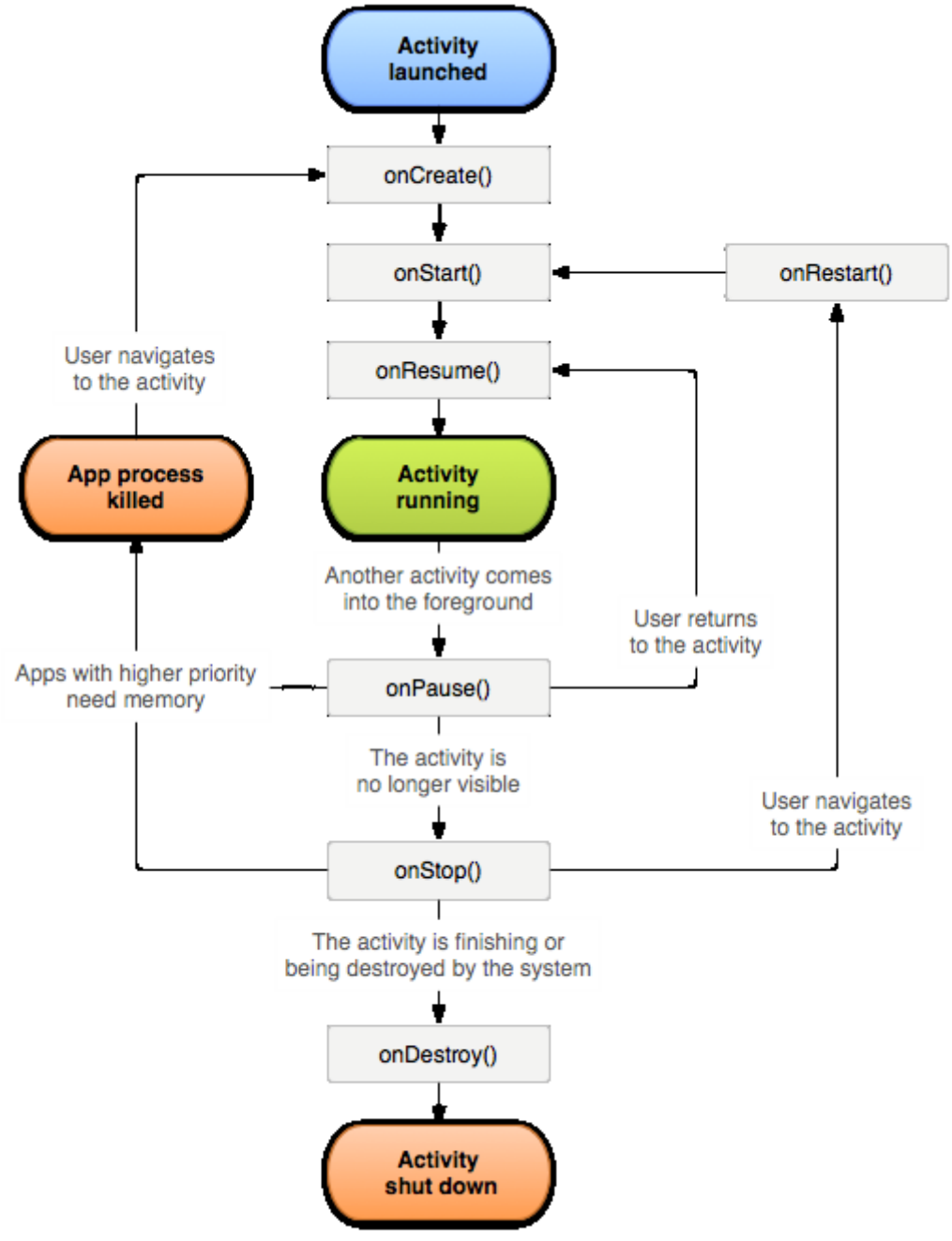
5.2.1. Obveščanje o prometnih dogodkih

Operacijski sistem Android uporablja aplikacijske komponente imenovane Activity, ki uporabniku predstavijo zaslon aplikacije, s katerim je uporabnik v interakciji. Preko teh komponent uporabnik upravlja aplikacijo, le-ta pa mu posreduje podatke. Aplikacija ima običajno glavno aplikacijsko komponento imenovano *main activity*, ki se prikaže uporabniku, ko prvič zažene aplikacijo. Vsak activity lahko zažene drug activity, da se izvede drugo dejanje. Vsakič, ko se zažene nov activity, se prejšnji ustavi, vendar ostane v aplikacijskem skladu. Aplikacijski sklad uporablja vrsto za upravljanje LIFO z shranjenimi activity objekti.

Activity objekt ima v grobem tri stanja: Resumed, Paused in Stopped. V stanju Resumed je activity objekt v ospredju in viden uporabniku. Tukaj lahko activity objekt izvaja različne klice metod na podlagi uporabniške interakcije. V stanju Paused je activity objekt takrat, ko je še vedno v ospredju, vendar ni več v centru – drug activity objekt je v centru, vendar je prejšnji še

vedno viden v ozadju. V stanju Stopped je activity objekt takrat, ko ga popolnoma prekrije drug activity objekt. Če je objekt v stanju Paused ali Stopped, lahko operacijski sistem objekt uniči [9].

Življenjski cikel Activity objekta je predstavljen na sledeči sliki (slika 16):

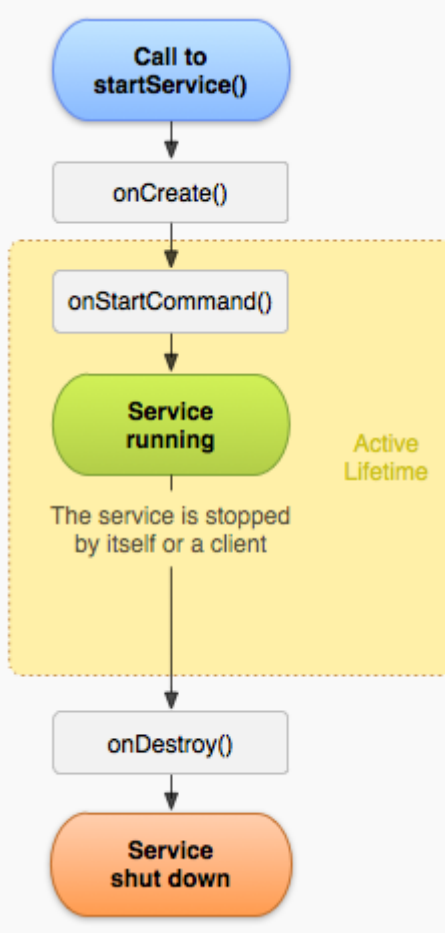


Slika 16: Življenski cikel Activity objekta

Na podlagi življenjskega cikla Activity objekta hitro ugotovimo, da ne moremo obveščati uporabnika o prometnih dogodkih, če aplikacija ni v centru, saj se celotno izvajanje aplikacije zamrzne. V ta namen smo uporabili Service aplikacijsko komponento oz. storitev. Storitev nam

omogoča, da izvajamo dolgotrajne operacije v ozadju, vendar ne ponuja uporabniškega vmesnika. Aplikacija lahko zažene storitev, ki bo tekla v ozadju, tudi če uporabnik uporablja druge aplikacije. Storitev lahko prav tako teče v ozadju, če aplikacija, ki jo je zagnala, umre. Na to moramo biti pozorni, saj ne želimo imeti storitve brez nadzora krovne aplikacije.

Življenjski cikel storitve (slika 17):



Slika 17: Življenski cikel storitve

V aplikaciji Moja Cesta definiramo storitev, ki bo opravljala sledeče naloge:

1. Pridobivala uporabnikovo trenutno lokacijo GPS
2. Spreminjala točke GPS v naslove ulic
3. Računala razdaljo uporabnikovega položaja do prometnih dogodkov
4. Obveščala uporabnika o bližnjih prometnih dogodkih
5. Komunicirala z aktivnostmi aplikacije

5.2.2. Pridobivanje trenutne lokacije uporabnika in spreminjanje koordinat GPS v naslove ulic

Pri razvoju aplikacije, ki se zaveda položaja uporabnika, lahko uporabimo dva načina pridobivanja lokacije – preko sprejemnika GPS, vgrajenega v napravo, ali preko mehanizma Android Network Location Provider, ki pridobi lokacijo preko omrežja. Čeprav je lokacijo pridobljena preko sprejemnika GPS veliko bolj natančna, prinaša nekaj pomanjkljivosti: velika poraba baterije, počasnost vračanja lokacije in deluje samo na prostem. Pridobivanje lokacije preko omrežja poteka preko triangulacije med baznimi postajami in zaznavanjem signalov obstoječih brezžičnih dostopnih točk. Ta način pridobivanja lokacije se odziva veliko hitreje in porabi manj baterije. Za pridobivanje lokacije lahko uporabimo oba načina vzporedno ali pa le enega od njih.

Pridobivanje uporabnikove lokacije je lahko neprijetno opravilo zaradi naslednjih faktorjev:

1. Več različnih virov lokacije –sprejemnik GPS, triangulacija med baznimi postajami in zaznavanje brezžičnih omrežij vrne kot rezultat več različnih lokacij. Potrebno se je odločiti, kateremu rezultatu zaupati in hkrati premisliti, kateri način je najbolj smiseln glede na porabo baterije, odzivnosti in natančnosti.
2. Uporabnik se premika – lokacijo je potrebno ponovno pridobiti na določen časovni interval zaradi možnih premikov uporabnika.
3. Natančnost rezultatov – različni izvori lokacije imajo različno natančnost. Potrebno se je odločiti, ali ima starejši lokacijski podatek večjo natančnost kot trenutni lokacijski podatek.

Pridobivanje lokacije v poteka s pomočjo objekta LocationManager in LocationListener. Objekt LocationManager od operacijskega sistema zahteva dostop do sistemskih virov za pridobivanje lokacije. Vse lokacijsko povezane dogodke nato obdela objekt LocationListener.

Primer kode za pridobivanje lokacije od obeh lokacijskih ponudnikov:

```
// Definiraj LocationManager
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);

// Definiraj poslušalca za lokacijske dogodke
LocationListener locationListener = new LocationListener() {
   public void onLocationChanged(Location location) {
     // Se kliče, ko pridobimo novo lokacijo
     makeUseOfNewLocation(location);
   }

   public void onStatusChanged(String provider, int status, Bundle extras) {}

   public void onProviderEnabled(String provider) {}

   public void onProviderDisabled(String provider) {}
 };
```

```
// LocationListener naj posluša za dogodke iz LocationManagerja
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0,
locationListener);
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);
```

V primeru uporabe obeh načinov pridobivanja lokacije, preko sprejemnika GPS in omrežja, potrebujemo mehanizem, ki vedno vzame boljšo lokacijo. Logika mehanizma je v osnovi sledeča:

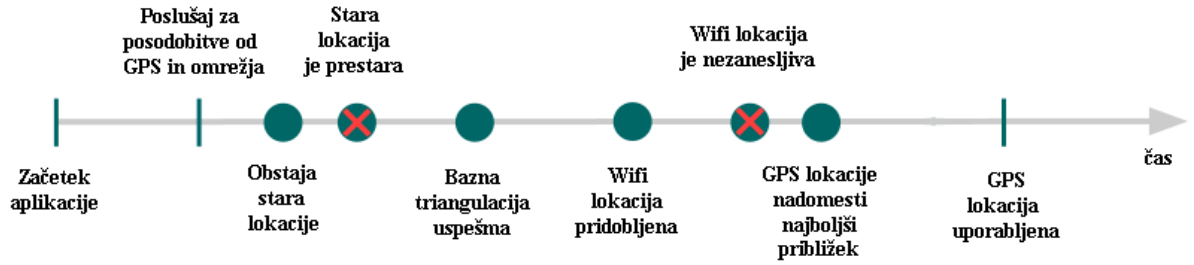
1. Preveri, če je nova lokacija bistveno novejša od stare.
2. Preveri, če je natančnost nove lokacije boljša od stare.
3. Preveri, iz katerega vira je lokacija in koliko lahko temu viru zaupamo.

Metodo za določanje boljše lokacije definiramo z naslednjim postopkom:

```
private boolean isBetterLocation(Location location, Location currentBestLocation) {
        if (currentBestLocation == null) {
                // Nova lokacije je vedno boljša, kot da ni lokacije
                return true;
        }

        // Preveri ali je nova lokacija starejša ali novejša
        long timeDelta = location.getTime() - currentBestLocation.getTime();
        boolean isSignificantlyNewer = timeDelta > TWO_MINUTES;
        boolean isSignificantlyOlder = timeDelta < -TWO_MINUTES;
        boolean isNewer = timeDelta > 0;

        // ČE je bilo več kot 2 minuti od zadnje znane lokacije uporabi novo lokacijo, ker se je
verjetno uporabnik premaknil
        if (isSignificantlyNewer) {
                return true;
                // Če je nova lokacija starejša od dveh minut, je verjetno slabša
        } else if (isSignificantlyOlder) {
                return false;
        }

        // Preveri, če je nova lokacija bolj ali manj natančna
        int accuracyDelta = (int) (location.getAccuracy() -
currentBestLocation.getAccuracy());
        boolean isLessAccurate = accuracyDelta > 0;
        boolean isMoreAccurate = accuracyDelta < 0;
        boolean isSignificantlyLessAccurate = accuracyDelta > 200;

        // Preveri če je nova lokacija pridobljena preko istega ponudnika lokacije kot stara
        boolean isFromSameProvider = isSameProvider(location.getProvider(),
currentBestLocation.getProvider());

        // Preveri kvaliteto lokacije z uporabo kombinacije časovnice in natančnosti
        if (isMoreAccurate) {
                return true;
        } else if (isNewer && !isLessAccurate) {
```

```
            return true;
        } else if (isNewer && !isSignificantlyLessAccurate && isFromSameProvider) {
            return true;
        }
        return false;
    }

    // Preveri, če gre zta isti ponudnik
    private boolean isSameProvider(String provider1, String provider2) {
        if (provider1 == null) {
            return provider2 == null;
        }
        return provider1.equals(provider2);
    }
```

Osnovni tok dogodkov za pridobivanje lokacije lahko uprizorimo na diagramu (slika 18):



Slika 18: Pridobivanje lokacije

Ob prejemu lokacijskih koordinat želimo uporabnika obvestiti tudi o ulici, na kateri se nahaja. Proces pridobivanja naslova iz koordinat GPS imenujemo obratni geokod in v Android aplikacijah poteka z uporabo objekta Geocoder.

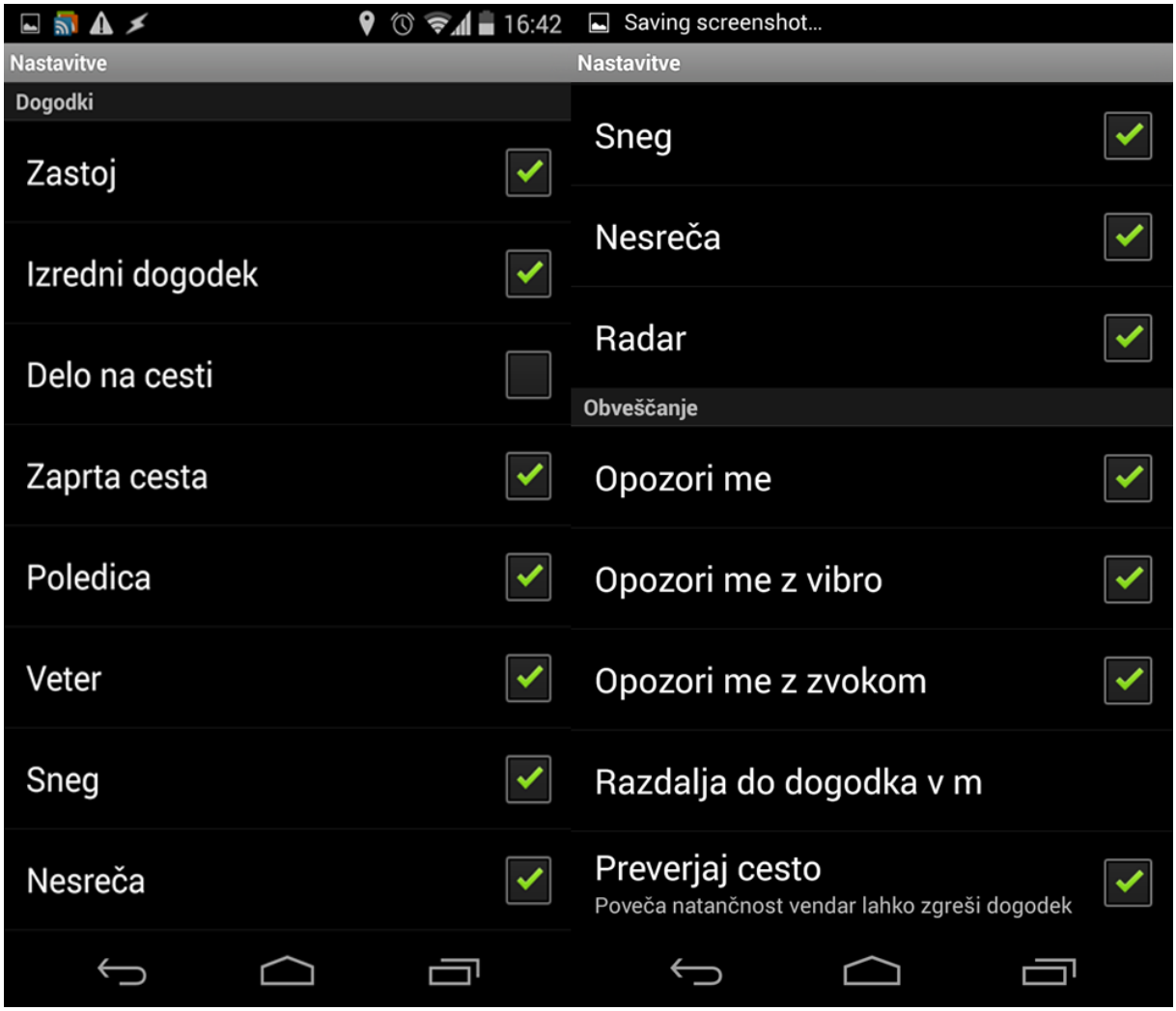
Primer pridobivanja naslova iz lokacije:

```
Geocoder geocoder = new Geocoder(getBaseContext());
private String geocode(Location location) {
        List<Address> addresses;
        String address = "";
        try {
            addresses = geocoder.getFromLocation(location.getLatitude(),
            location.getLongitude(), 1);
            if (addresses != null) {
                address = addresses.get(0).getAddressLine(0);
                if (address.length() < 1) {
                    address = getBaseContext().getString("Unknown address");
                }
            }
        } catch (IOException e) {
            address = currentAddress;
            Log.d(TAG, "Error when reverse geocoding");
        }
        Log.d(TAG, "Current address: " + address);
        return address;
    }
```

5.2.3. Obveščanje o dogodkih

Uporabniku smo želeli omogočiti spreminjanje nastavitev obveščanja. Želimo, da si uporabnik sam nastavi dogodke, ki ga zanimajo, in kako ga naj aplikacija opozori. Izbira lahko med opozarjanjem z vibriranjem in zvokom, na kolikšni razdalji želi prejeti opozorilo in če naj aplikacija zaradi večje natančnosti določanja možnosti zadetka dogodka upošteva tudi ulico, na kateri se uporabnik nahaja (slika 19).

Vse spremembe obveščanja, ki jih uporabnik vnese, se shranijo na napravi in pošljejo na strežnik, da se lahko odražajo v tudi spletni aplikaciji.

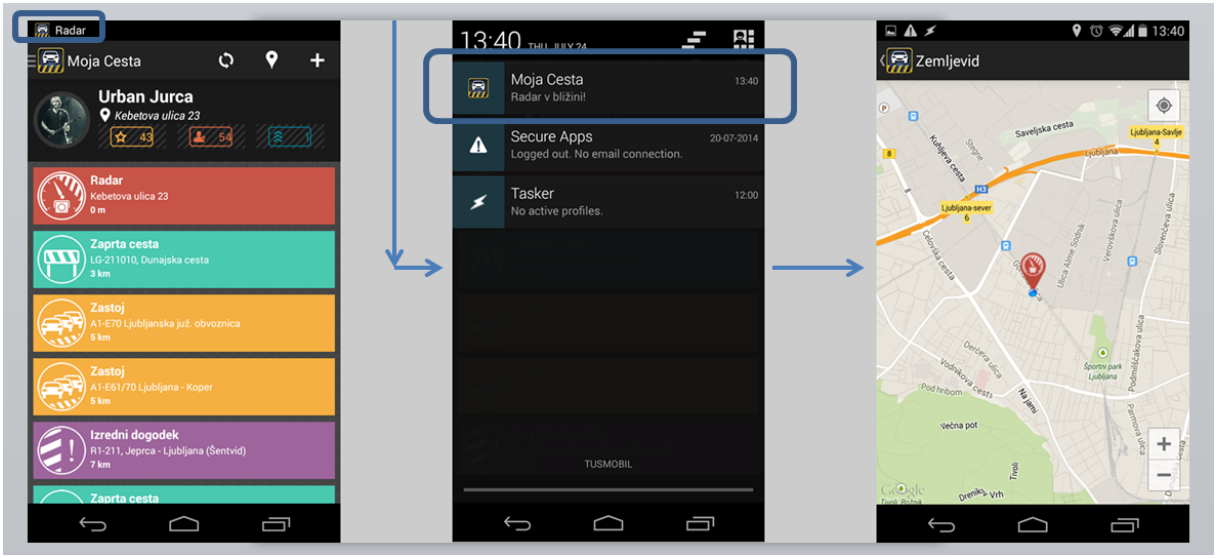


Slika 19: Spreminjaje obveščanja

Algoritem za opozarjanje deluje po naslednjih načelih:

1. Ko se lokacija posodobi, zaženi funkcijo opozarjanja.
2. Če je opozarjanje izklopljeno, ne naredi ničesar, sicer pa preveri, kolikšna je nastavljena razdalja za opozarjanje. Sprehodi se čez celotni seznam dogodkov, na katere je naročen uporabnik, in če se kateri od dogodkov nahaja na določeni razdalji od uporabnika, nadaljuje na naslednji korak, sicer konča.
3. Preveri, ali je uporabnik omogočil funkcijo preverjanja ceste. Če je funkcija vklopljena, aplikacija preveri, če se vsaj dve besedi ulice dogodka in trenutne ulice ujemata, ali pa se morda ujema celotna ulica. V primeru da se, obstaja velika verjetnost, da se uporabnik nahaja na isti ulici in aplikacija opozori uporabnika. Če funkcija ni vklopljena, aplikacija o tem opozori uporabnika.

Privzete nastavitve uporabnika obvestijo z vibriranjem in zvokom, ko se ta približa dogodku na 500 m. Pojavi se tudi sporočilo v zgornji vrstici zaslona s kratkim opisom dogodka. Ko uporabnik opravilno vrstico razširi z vertikalnim gibom navzdol, vidi bolj podroben opis opozorila. S klikom na opozorilo se zažene aplikacija Moja Cesta in uporabniku ponudi zaslonsko masko z zemljevidom, njegovo trenutno lokacijo in lokacijo dogodka (slika 20).



Slika 20: Obveščanje uporabnika

Kot je bilo omenjeno, poteka pridobivanje lokacije in računanje oddaljenosti do dogodkov v ozadju; zato potrebujemo mehanizem, preko katerega lahko prikažemo rezultate uporabniku, ko ima odprto aplikacijo. Načinov, preko katerih lahko to naredimo, je več, vendar smo se odločili za uporabo razpršenega oddajanja sporočil Broadcasts. V procesu, ki želi poslati sporočilo (v našem primeru je to storitev, ki teče v ozadju), definiramo podatkovno strukturo s pomočjo objekta Intent in ga pošljemo. Definiramo sporočilo o novi lokaciji. Uporabimo ga za posodabljanje uporabniškega vmesnika aplikacije. Ob spremembi lokacije posodobimo izpis ulice, na kateri se nahajamo, in preuredimo seznam dogodkov od najbližjega do najbolj

oddaljenega. V sporočilo zapakiramo koordinate GPS in ime ulice. Ker gre za razpršeno sporočilo, ga lahko slišijo vse tekoče aplikacije na napravi in hkrati lahko naša aplikacija sliši vsa ostala razpršena sporočila. Da se zavarujemo pred napačnimi sporočili, definiramo akcijo, ki ji vpišemo v podatkovno strukturo. V aplikaciji moramo definirati sprejemnik, ki bo poslušalo samo naša sporočila, in akcije, ki naj se izvedejo [7], [8].

Primer komunikacije med storitvijo in aplikacijo ponazorimo na dveh primerih kode. Primer razpršenega sporočila:

```
private void broadcastLocationChange() {
        broadcastIntent = new Intent();
        broadcastIntent.setAction("UPDATE_LOCATION");
        broadcastIntent.putExtra("LOCATION_DATA", currentLocation.getData());
        getBaseContext().sendBroadcast(broadcastIntent);
}
```

Primer sprejemnika:

```
//Naredi filter sporočil
IntentFilter locationFilter = new IntentFilter();
//Ujemi samo sporočila z akcijo UPDATE_LOCATION
locationFilter.addAction("UPDATE_LOCATION");
//Poveži filter s sprejemnikom
registerReceiver(locationReceiver, locationFilter);
private BroadcastReceiver locationReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
                //Tukaj vpišemo akcije
        }
};
```

5.2.4. Dodajanje dogodkov

Mobilno aplikacijo smo želeli nadgraditi tudi s funkcionalnostjo dodajanja dogodkov. Tako smo hoteli motivirati uporabnike aplikacije, da postanejo poročevalci prometnih dogodkov. S tem dosežemo boljšo ažurnost podatkov, kot jo ponudi Dars. Za ta namen smo razvili posebno okno aplikacije, preko katerega lahko uporabnik z enim klikom pošlje dogodek (slika 21).

Slika 21: Dodajanje prometnih dogodkov

Uporabnik s klikom na gumb »+« iz glavnega zaslona aplikacije preide na zaslon, kjer lahko poroča o prometnih dogodkih. Podana mu je maska z devetimi znaki, ki predstavljajo prometne dogodke:

1. Znak  predstavlja dogodek **Zastoj.**
2. Znak  predstavlja dogodek **Izredni dogodek**.
3. Znak  predstavlja dogodek **Delo na cesti**.
4. Znak  predstavlja dogodek **Zaprta cesta**.
5. Znak  predstavlja dogodek **Poledica**.
6. Znak  predstavlja dogodek **Veter**.
7. Znak  predstavlja dogodek **Sneg**.
8. Znak  predstavlja dogodek **Nesreča**.

9. Znak  predstavlja dogodek **Radar**.

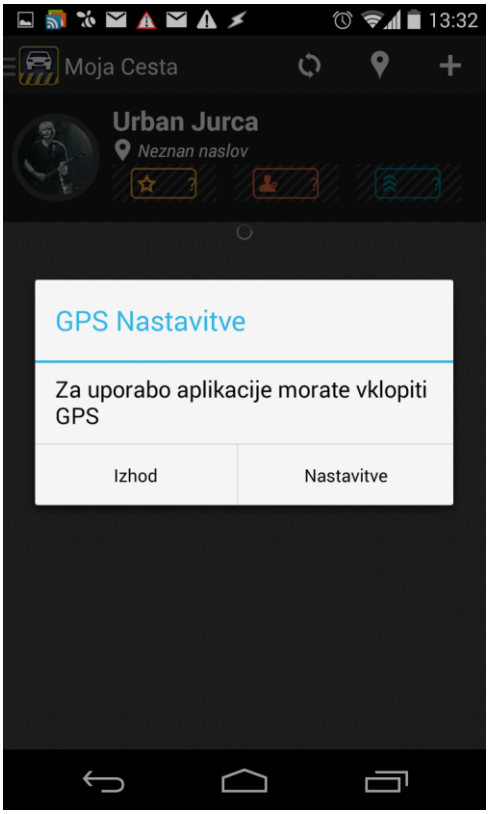
Ob kliku na gumb se na strežnik pošlje informacija o uporabniku, dogodku, trenutnih lokacijskih koordinatah in trenutni naslov. Aplikacija nato uporabnika vrne nazaj na glavno zaslonsko masko.

5.2.5. Asinhrono upravljanje podatkov in varnost

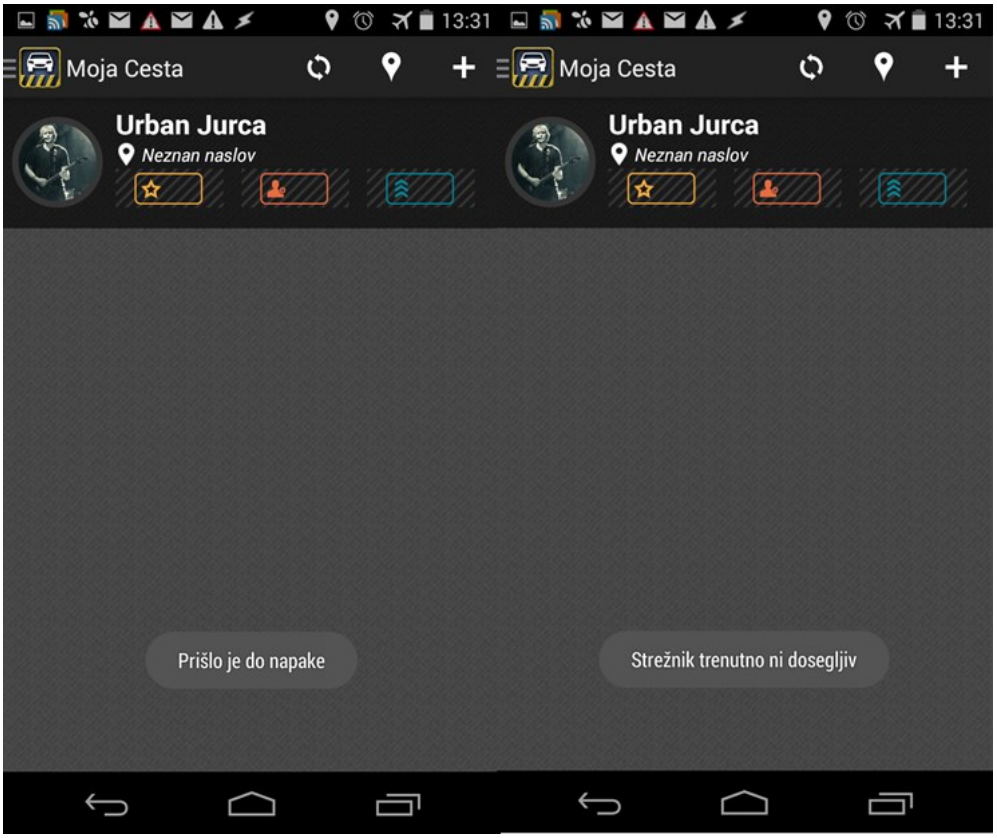
Zavoljo boljše uporabniške izkušnje je ključnega pomena za mobilno aplikacijo asinhrono upravljanje s podatki. Dolge operacije, kot so pridobivanje in pošiljanje podatkov preko internetnih povezav, je zaželeno izvajati izven glavne niti delovanja aplikacije, saj lahko potencialno ustavijo delovanje vizualnih komponent aplikacije in posledično pokvarijo uporabniško izkušnjo. Zaradi tega smo vse take operacije spremenili v asinhrone operacije. Potrebno je bilo zagotoviti tudi mehanizme, ki omogočajo lovljenje izrednih dogodkov, kot so trenutna nedosegljivost spletnega strežnika, prekinitev povezave med prenosom podatkov in ostale izredne dogodke. V takih primerih mora aplikacija ponovno nadaljevati prenos podatkov, ko je to le mogoče. V nasprotnem primeru lahko pride do neobravnavanih izjem in se delovanje aplikacije potencialno ustavi.

Uvedli smo tudi preverjanje nastavitev naprave z namenom popolne omogočitve funkcij aplikacije. Tako ob zagonu preverjamo, ali ima uporabnik omogočene lokacijske storitve in primerno uporabnika pozovemo, da naj to funkcionalnost naprave vklopi (slika 22). Brez lokacijskih storitev ne moremo uporabljati aplikacije.

V primeru, da uporabnikova naprava trenutno nima dostopa do interneta ali pa je prišlo pri povezovanju do strežnika do napake, ga temu primerno obvestimo (slika 23). Izvajanje aplikacije se v nobenem primeru ne sme ustaviti. Za razvijalce je pomembno, da se aplikacija preveri v vseh možnih ekstremnih primerih. Predvsem je pomembno zbirati informacije testnih uporabnikov, saj lahko pridejo do rezultatov in situacij, ki si jih razvijalci ne bi mogli zamisliti.



Slika 22: Preverjanje lokacijskih nastavitev



Slika 23: Obveščanje o napakah

6. Možne razširitve

Na podlagi testiranj in zbiranj povratnih informacij o spletni in mobilni aplikaciji bi lahko naredili nekaj izboljšav. Trenutno ima spletna aplikacija manj funkcionalnosti kot mobilna. Spletni aplikaciji bi bilo smiselno dodati možnost dodajanja dogodkov v bazo obstoječih dogodkov. Dodali bi lahko tudi možnost dodatnega opozarjanja v spletni aplikaciji glede na zaznano lokacijo. Tako bi obe platformi bolj poenotili. Zaradi velike fragmentacije platforme Android bi bilo potrebno aplikacijo Moja Cesta testirati še na več različnih Android napravah. Aplikacijo bi bilo smiselno razširiti tudi na platformo iOS in Windows Phone in s tem pokriti vse priljubljene platforme.

Trenutno v mobilni aplikaciji spremljamo samo absolutno razdaljo do dogodkov. Smiselno bi bilo spremeniti absolutno razdaljo v čas, v katerem dogodek dosežemo. Tako bi nas aplikacija opozorila na prihajajoč prometni dogodek pravočasno, ne glede na hitrost naše vožnje. Hitrost premikanja bi lahko izračunali preko lokacijskih premikov in iz tega dobili povprečen čas do dogodka. To bi izboljšalo uporabniško izkušnjo. Mobilno aplikacijo bi lahko nadgradili tudi tako, da bi si na podlagi prepotovanih poti zapomnila naše redne poti in nas obveščala samo o dogodkih na naši navadni poti. To funkcionalnost bi lahko nadgradili tako, da bi uporabnik lahko definiral še časovne intervale prihoda ali odhoda npr. v službo in bi ga aplikacija že pred pričetkom potovanja obvestila o prometnih dogodkih na poti.

7. Zaključek

Cilj diplomskega dela je bil osvojiti veščine Android programiranja ter ogrodja Laravel PHP, kot tudi razviti aplikacijo Moja Cesta, ki pomaga voznikom na cesti s sprotnim in ažurnim obveščanjem o stanju na cestah. Izdelana aplikacija prikazuje vse za uporabnika pomembne podatke o aktualnih prometnih dogodkih in dogodkih, ki jih generirajo uporabniki aplikacije. Razvili smo aplikacijo, ki uporabnikom nudi možnost tekmovanja s souporabniki; s tem smo želeli povečati možnost posvojitve in razširitve aplikacije. Uporabili smo vse tehnologije, ki smo si jih zadali preveriti, in smo jih uspešno implementirali v aplikacijo. Ob primerni dodelavi aplikacije z omenjenimi razširitvami, ob primernem testiranju in nadaljnjem razvoju bi aplikacijo naložili na aplikacijsko trgovino. Tako bi jo javno distribuirali za uporabo in pomoč voznikom.

Spletna aplikacija je objavljena na naslovu http://www.mojacesta.com.

8. Viri

[1] W3, Geolocation API Specification
http://dev.w3.org/geo/api/spec-source.html

[2] Google, Google + platform
https://developers.google.com/+/

[3] Stackoverflow, Are the PUT, DELETE, HEAD, etc methods available in most web browsers?
https://stackoverflow.com/questions/165779/are-the-put-delete-head-etc-methods-available-in-most-web-browsers

[4] Google, Google + API
https://developers.google.com/+/api/

[5] GPS Humano, MySQL DATETIME vs TIMESTAMP vs INT performance and benchmarking with MyISAM
http://gpshumano.blogs.dri.pt/2009/07/06/mysql-datetime-vs-timestamp-vs-int-performance-and-benchmarking-with-myisam/

[6] Statistični urad republike Slovenije, Transport
http://www.stat.si/tema_ekonomsko_transport.asp

[7] J. Steele, N. To, The Android Developer's Cookbook: Building Applications with the Android SDK, New Jersey: Addison-Wesley Educational Publishers Inc, 2010

[8] R. Meier, Professional Android 4 Application Development, John Wiley & Sons Inc, 2012

[9] W. Lee, Beginning Android 4 Application Development, John Wiley & Sons Inc, 2012

[10] Laravel, Laravel documentation
http://laravel.com/docs

[11] Composer, Composer documentation
https://getcomposer.org/doc/

[12] Google maps
https://developers.google.com/maps/

[13] ApiGee, ApiGee WebApi design
https://pages.apigee.com/web-api-design-website-h-ebook-registration.html?utm_source=resources-main&utm_medium=website&utm_campaign=ebook&utm_content=api-design